Developing Browser-Based Software Systems for Medical Image Processing and Radiological Reporting

PhD dissertation

Dr. Ahmed Mouhamad Harmouche

Clinical Neurosciences Doctoral School (D221)

Imaging in neuroscience (B-3/2014)

Leader of the Doctoral School: Prof. Dr. Sámuel Komoly

Program leader: Prof. Dr. Péter Bogner

Supervisor: Dr. Arnold Tóth

University of Pécs Medical School

Department of Medical Imaging

Pécs, 2023

Table of Contents

1. List of abbreviations

ACR – American College of Radiology

AI – Artificial Intelligence

BET – Brain Extraction Tool

CNN – Convolutional Neural Network

CSS – Cascading Style Sheets

CT – Computed Tomography

DICOM – Digital Imaging and Communications in Medicine

DOM – Document Object Model

DSL – Domain Specific Language

DTI – Diffusion Tensor Imaging

FAST – FMRIB’s Automated Segmentation Tool

FDT – FMRIB’s Diffusion Toolbox

FLIRT – FMRIB's Linear Image Registration Tool

fMRI – functional Magnetic Resonance Imaging

FEAT – FMRIB's Expert Analysis Tool

FIRST – FMRIB's Integrated Registration and Segmentation Tool

FNIRT – FMRIB’s Non-Linear Image Registration Tool

FSL – FMRIB Software Library

GPU – Graphical Processing Unit

HIS – Hospital Information System

HL7 – Health Level 7

HTML – HyperText Markup Language

HTML5 – HyperText Markup Language 5

ITK – Insight Segmentation and Registration Toolkit

JSON – JavaScript Object Notation

LLM – Large Language Model

LLVM – Low Level Virtual Machine

MINC – Medical Imaging NetCDF

MRI – Magnetic Resonance Imaging

NEMA – National Electrical Manufacturers Association

NIfTI – Neuroimaging Informatics Technology Initiative

PACS – Picture Archiving and Communication System

PET – Positron Emission Tomography

RIS – Radiology Information System

SDK – Software Development Kit

SPA – Single Page Application

SPM – Statistical Parametric Mapping

SWI – Susceptibility Weighted Imaging

TNM – Tumor, Node, Metastasis

TPU – Tensor Processing Unit

URL – Uniform Resource Locator

VBM – Voxel Based Morphometry

WebGL – Web Graphics Library

2. Introduction

The modern era of digital medicine has witnessed incredible pace in technological development, enhancing the ability of healthcare professionals to provide better patient care [1]. One critical area where these advancements have been most apparent is medical imaging and radiological reporting, which plays a vital role in diagnosing, treating, and managing numerous health conditions [2]. The development of browser-based software systems for these fields is, therefore, not only significant but also transformative in its implications for global health [3].

It is increasingly clear that medical image processing and radiological reporting is moving from being predominantly hospital-based to more decentralized, user-friendly, and accessible systems [4]. In the past, the use of these systems was often restricted by the need for high computational power, proprietary software, and complex user interfaces. However, advancements in computing technology [5] and the growth of cloud-based services have provided an opportunity to revolutionize these systems' delivery and access.

Browser-based software systems are at the forefront of this evolution [6] [7]. They provide the advantage of platform-independence, enabling healthcare professionals to access medical images and reports from virtually any device with an internet connection. This flexibility promotes better patient care by enabling faster, more efficient diagnoses and more effective communication among healthcare professionals.

Moreover, the complexity and steep learning curve associated with traditional medical software systems often pose a challenge for medical professionals. Ease-of-use is a key attribute of any software system, and this holds especially true for those used in healthcare. These professionals must navigate complex data quickly, and their tools should be intuitive and straightforward, reducing time spent on technicalities and freeing up more time for patient care.

The browser-based software systems designed for medical image processing and radiological reporting will streamline the way healthcare providers access, interpret, and share medical images and reports. By simplifying the user interface and improving accessibility, these systems can minimize technical barriers, enhance efficiency, and ultimately contribute to improved patient outcomes.

2.1. An overview of radiological information systems

Radiological Information Systems (RISs) have emerged as indispensable components in managing healthcare imaging departments, as they streamline administrative and operational tasks, enhance workflow efficiency, and facilitate communication among healthcare professionals. RISs are specialized software systems designed to address the multifaceted needs of radiology departments, incorporating various functions, such as patient scheduling, examination tracking, reporting, and billing. By integrating RISs with other information systems like Picture Archiving and Communication Systems (PACSs) [8] and Hospital Information Systems (HISs) [9], seamless information exchange is enabled, thereby improving the overall quality of patient care.

A RIS generally consits of several core components that work together to manage the different aspects of radiology department operations:

1. **Patient Registration and Scheduling:**

   This module manages patient demographic information, examination appointments, and referral details. It enables efficient scheduling of radiological examinations, taking into account resource availability and patient preferences.

2. **Examination Tracking and Workflow Management:**

   This component monitors the progress of radiological examinations throughout their lifecycle, from the initial request to the final report delivery. It also facilitates the optimization of workflows by identifying bottlenecks, managing resources, and ensuring seamless coordination among healthcare professionals.

3. **Radiological Reporting:**

   A crucial element of RIS, the reporting module enables radiologists to generate, store, and distribute diagnostic reports. This module often includes tools for dictation,

transcription, report editing, and electronic signature, as well as access to standardized templates and terminology resources.

4. **Image and Data Management:**
   RIS often interface with PACS to handle the storage, retrieval, and distribution of medical images and associated data. This integration ensures that radiologists can easily access relevant images and clinical information while creating diagnostic reports.

5. **Quality Assurance and Performance Monitoring**:
   This component of RIS allows radiology departments to monitor and evaluate the quality of their services by tracking key performance indicators, such as report turnaround time, examination accuracy, and patient satisfaction.

6. **Billing and Financial Management:**
   RIS include functionality for managing the financial aspects of radiology services, such as generating invoices, processing insurance claims, and tracking accounts receivable.

Modern RISs are designed to be highly functional, user-friendly, and interoperable, ensuring seamless integration with other healthcare information systems. Interoperability is achieved through the use of standardized data exchange protocols, such as Digital Imaging and Communications in Medicine (DICOM) [10] for image data and Health Level 7 (HL7) [11] for clinical and administrative information.

2.2. Basics of medical image processing

Medical image processing encompasses a wide array of techniques and algorithms aimed at enhancing, analyzing, and interpreting medical images to assist healthcare professionals in diagnosing and treating diseases. These techniques can involve various operations, such as image filtering, segmentation, registration, and reconstruction, which may be applied to

different types of medical images, including computed tomography (CT), magnetic resonance imaging (MRI), and ultrasound.

2.2.1. The DICOM format

The DICOM format is a widely adopted standard for the storage, exchange, and management of medical images and associated metadata. Developed by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA), DICOM ensures interoperability among various medical imaging devices, software applications, and information systems, facilitating seamless integration within healthcare environments. DICOM images comprise two main components: the image data, which typically represents a two-dimensional array of pixel values, and the header, which contains essential metadata about the image, such as patient demographics, acquisition parameters, and modality-specific information. The DICOM format supports various image compression schemes, enabling efficient storage and transmission of medical images without significant loss of quality.

2.2.2. Medical image processing applications

Several software applications and libraries have been developed for processing and analyzing medical images, in both clinical and research settings. Some popular software tools include the FMRIB Software Library (FSL) [12], an open-source software suite for the analysis of functional, structural, and diffusion MRI data, which includes tools for image preprocessing, statistical analysis, and visualization. Another widely used application is 3D Slicer [13], an open-source, extensible platform for medical image processing and visualization that supports a wide range of image formats, including DICOM. It offers various features, such as image segmentation, registration, and three-dimensional visualization, and can be customized through plugins.

In addition to FSL and 3D Slicer, the ITK (Insight Segmentation and Registration Toolkit) [14] is an open-source, cross-platform C++ library for image processing, segmentation, and registration. It provides a comprehensive collection of algorithms for medical image analysis and is widely used in both academia and industry. For radiology workflows specifically,

OsiriX [15] is a DICOM viewer and image processing software for macOS that supports various image manipulation and analysis features, such as multiplanar reconstruction, volume rendering, and image fusion.

2.3. Neuroimaging

2.3.1. Description

Neuroimaging is a subset of medical image processing that focuses on the visualization and analysis of the structure and function of the brain and central nervous system. It encompasses a wide range of imaging modalities, such as magnetic resonance imaging (MRI), functional magnetic resonance imaging (fMRI), diffusion tensor imaging (DTI), positron emission tomography (PET), and computed tomography (CT), among others. Neuroimaging plays a crucial role in clinical diagnosis, treatment planning, and research related to various neurological and psychiatric disorders, as well as the understanding of normal brain function.
MRI is a particularly important modality in neuroimaging, as it is noninvasive and provides high-resolution images of the brain's anatomy without exposing patients to ionizing radiation. MRI relies on the principles of nuclear magnetic resonance to generate images based on the relaxation properties of hydrogen nuclei within tissue. By applying different sequences of radiofrequency pulses and magnetic field gradients, MRI can produce images with varying contrast and sensitivity to specific tissue characteristics, such as T1 and T2 relaxation times, proton density, and diffusion properties.

2.3.2. File formats

In neuroimaging, data is often stored in file formats specifically designed for this purpose. The Neuroimaging Informatics Technology Initiative (NIfTI) [16] format is one such widely adopted standard for storing and analyzing MRI, fMRI, CT, and any other neuroimaging data. The NIfTI format extends the widely used Analyze [17] format by incorporating additional header information, such as image orientation, spatial and temporal units, and data scaling, which is essential for proper interpretation and analysis of neuroimaging data.

Another commonly used format is the Medical Imaging NetCDF (MINC) format, developed by the Montreal Neurological Institute, which supports multidimensional, multi-variate, and multi-resolution data, along with extensive metadata.

2.3.3. Common processing algorithms

MR image processing involves several challenges related to the complexity and variability of brain anatomy, as well as the presence of noise, artifacts, and distortions in the acquired images. Common preprocessing steps include image registration (Fig. 1.), which involves aligning images from different time points or modalities; image segmentation, which involves partitioning the image into distinct regions corresponding to different tissue types or anatomical structures; and image denoising, which aims to reduce noise and enhance image quality. A subset of image segmentation is brain extraction (Fig. 2.), where the non-brain tissue is separated from the brain tissue, and is removed from the image. In addition to these techniques, MRI data can be analyzed using various quantitative methods, such as voxel-based morphometry (VBM) for studying structural differences between groups or individuals, and functional connectivity analysis for exploring functional interactions between brain regions.

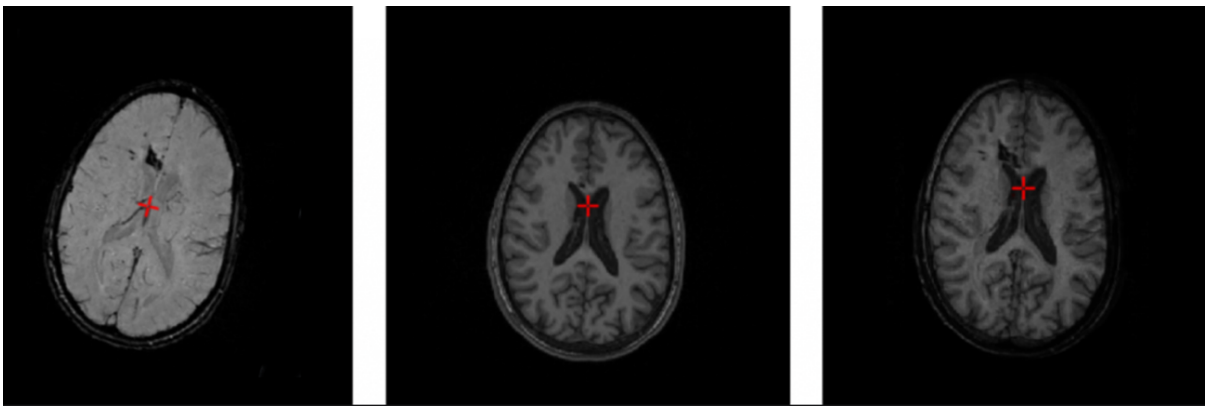

Figure 1 – Linear registration – In this figure three axial slices of three different MRI volumes are shown. The left image is a slice of an SWI MRI volume, the middle image is a slice of a T1-weighted MRI volume, and the right image shows a slice of a volume that is the combination of the first two. The FMRIB's Linear Image Registration Tool (FLIRT) was used to register the SWI volume to the reference T1 volume, which were then blended together to produce the combined volume shown on the right image. The red cross is used to navigate the slices of the volume, and to select voxels. The images are screenshots from our WebMRI application. (To better illustrate the correction the linear registration performs, the first image was rotated clockwise, thus the red cross is also rotated.)

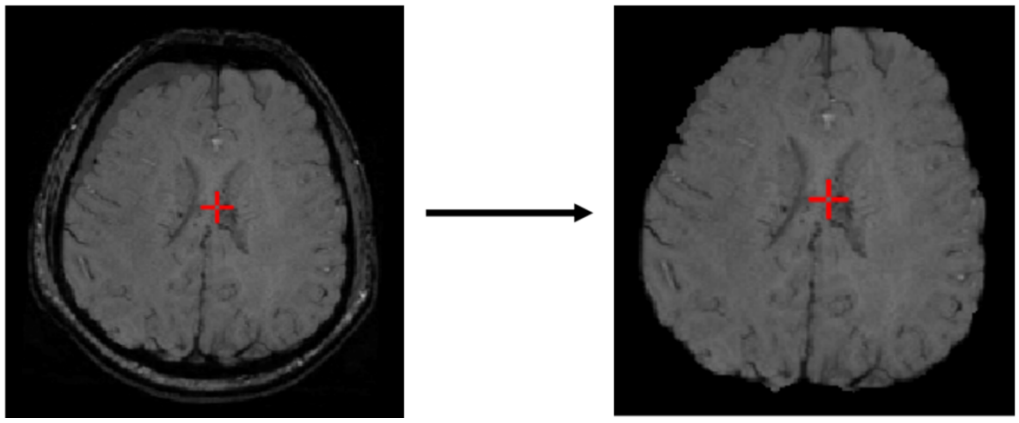

Figure 2 – Brain extraction - The left image shows an axial slice of an SWI MRI volume, and the right image shows the same slice after the FSL Brain Extraction Tool (BET) has been applied to the volume. The red cross is used to navigate the slices of the volume, and to select voxels. The images are screenshots from our WebMRI application.

Several software tools have been developed specifically for neuroimaging data processing and analysis. One of the most widely used tools is the aformentioned FSL. It includes tools for image registration (FLIRT, FNIRT) [18] [19], segmentation (FAST, FIRST), statistical analysis (FEAT, RANDOMISE), brain extraction (BET) [20], and diffusion modeling (FDT), among others.

Another popular software package is the Statistical Parametric Mapping (SPM) software, developed by the Wellcome Trust Centre for Neuroimaging. SPM is an open-source MATLAB-based toolbox for the analysis of functional and structural neuroimaging data, offering a wide range of preprocessing, statistical modeling, and visualization tools.

2.4. Radiological reporting

As shown earlier in the Introduction, radiological reporting is an important component of the RIS, and in the diagnostic process of medical imaging, as it communicates the interpretation of the radiologist of the imaging findings to the referring physicians and other healthcare providers. Accurate, clear, and concise reporting is essential for effective patient care and treatment planning. Two main approaches to radiological reporting currently dominate the field: dictation-based reporting and template-based structured reporting.

Dictation-based reporting is the most commonly used method for generating radiology reports. In this approach, the radiologist verbally describes the imaging findings and their clinical implications using a microphone, and the recorded speech is either transcribed by a medical transcriptionist or converted to text using speech recognition software [21]. Dictation-based reporting allows for free-form, natural language expression, which can be efficient and flexible, enabling radiologists to tailor their descriptions to the specific case at hand. However, this method can also result in significant variability in report structure, terminology, and content, as different radiologists may use different styles, abbreviations, and levels of detail, even when describing the same findings. This variability can lead to challenges in interpreting and comparing reports across different readers, institutions, or time points, potentially affecting the quality and consistency of patient care.

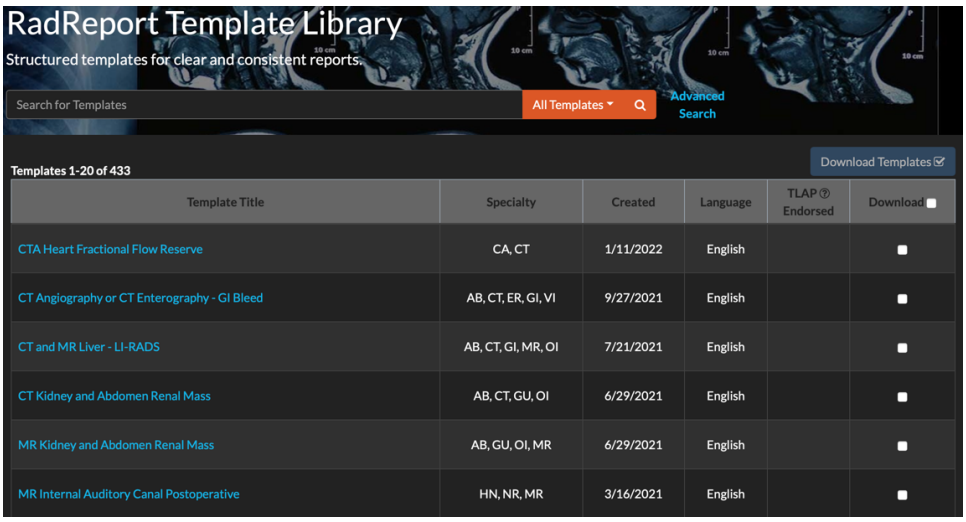

Figure 3 – A list of structured reporting templates as seen on the RadReport Template Library website of the Radiological Society of North America (RSNA). The templates can be searched and filtered based on specialty (CT, MR, etc.), date of creation and language, among other parameters. Selecting a template from the list navigates to a page that renders it. Source: https://radreport.org

Template-based structured reporting [22] aims to address these limitations by providing a standardized framework for organizing and presenting radiological findings. Structured reporting templates consist of predefined sections, headings, and data elements that guide the radiologist in systematically describing the relevant imaging findings, using consistent terminology and layout. The use of structured reporting templates can help improve the clarity, completeness, and consistency of radiology reports [23], making it easier for referring physicians to extract the pertinent information and reducing the risk of miscommunication or misinterpretation.

Structured reporting also has several additional advantages over dictation-based reporting. It facilitates the extraction of structured data from reports, which can be used for quality assurance, research, and decision support purposes, as well as the development of radiology informatics tools, such as natural language processing and machine learning algorithms. Furthermore, structured reporting can improve report turnaround time and reduce transcription errors, by directly entering the relevant information into the electronic medical record.

However, the adoption of structured reporting also faces certain challenges and drawbacks. Some radiologists may find the use of templates restrictive or time-consuming, especially if the templates are not well-designed or tailored to their specific workflow and clinical context. Moreover, the implementation of structured reporting may require significant resources for template development [24] [25], customization, and integration with existing radiology information systems, as well as training and support for radiologists in using the new reporting tools.

In conclusion, both dictation-based and template-based structured reporting have their respective benefits and drawbacks in the context of radiological reporting. The choice between these approaches depends on various factors, including the specific needs and priorities of the radiology department, the available resources and infrastructure, and the preferences of the radiologists themselves. Ultimately, the goal should be to adopt reporting practices that optimize the quality, efficiency, and consistency of radiological communication, in order to support the best possible patient care and outcomes.

2.5. Web-based software systems in radiology

The advent of web-based technologies and the increasing capabilities of modern web browsers have had a significant impact on various aspects of radiology, including image viewing, processing, and reporting. Web-based software systems offer a number of advantages over traditional desktop applications, such as platform independence, ease of deployment and maintenance, and the ability to access data and services from any device with an internet connection.

One of the main applications of web-based software systems in radiology is the development of web-based PACSs. PACSs are essential components of modern radiology departments, as they provide the means to store, retrieve, and distribute medical images across different modalities, workstations, and facilities. Web-based PACSs allow radiologists and other healthcare professionals to access and view medical images through a standard web browser, without the need for specialized software or hardware. This can improve the efficiency and flexibility of image interpretation, as well as facilitate collaboration and consultation among clinicians and radiologists, regardless of their physical location.

Several web-based PACS solutions have been developed and commercially available, such as Ambra Health, and Sectra PACS, among others. These systems typically leverage web technologies, such as HTML5, JavaScript, and WebGL, to provide advanced image viewing and manipulation capabilities within the browser, including multi-planar reconstruction, window leveling, and 3D visualization. Some web-based PACS also incorporate additional features, such as image annotation, reporting, and integration with electronic medical records, to support the end-to-end radiology workflow.

Another area where web-based software systems are making an impact is in the field of medical image processing. Traditionally, medical image processing tools have been developed as standalone desktop applications or libraries, often requiring significant computational resources and specific software dependencies. However, recent advances in web technologies, such as WebAssembly and Web Workers, have enabled the development of browser-based image processing applications that can perform complex operations, such as segmentation, registration, and feature extraction, directly within the browser, without the need for server-side processing or software installation.

A notable example of a web-based medical image processing platform is BrainBrowser, an open-source JavaScript library that provides 2D and 3D visualization of neuroimaging data,

including MRI, fMRI, and DTI, in the browser. BrainBrowser leverages WebGL for hardware-accelerated rendering and supports various neuroimaging file formats, such as NIfTI and MINC. Thanks to its modular nature, it's easily extensible.
Another extensible web-based solution is cornerstone.js, which is a set of well-defined DICOM and image processing libraries. Using these components one can develop a full-fledged DICOM image viewer. From annotations to windowing, the feature set is rich enough to cover most use-cases.

2.6. Artificial neural networks in radiology

The growing capabilities of Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs) accelerated the field of Artificial Intelligence (AI). With the increasing processing power of these devices more sophisticated and larger neural networks could be developed, and they could be trained on bigger datasets in shorter amount of time. Since radiological devices, such as a CT scanner or an MRI machine, produce large amounts of data even for a single patient, GPU/TPU innovations benefited the field of medical AI greatly. From segmenting organs on a whole-body CT scan [26], to detecting pneumonia on an X-Ray image [27], the applications of neural networks in radiology are highly diverse. For image processing use cases, Convolutional Neural Networks (CNN) are widely used (Fig. 4.). The inspiration to CNNs came from the neural network architecture of the animal visual cortex. In the visual cortext a single neuron responds to stimuli coming from a restricted region of the visual field. These regions are called receptive fields. These fields partially overlap to cover the whole visual field.

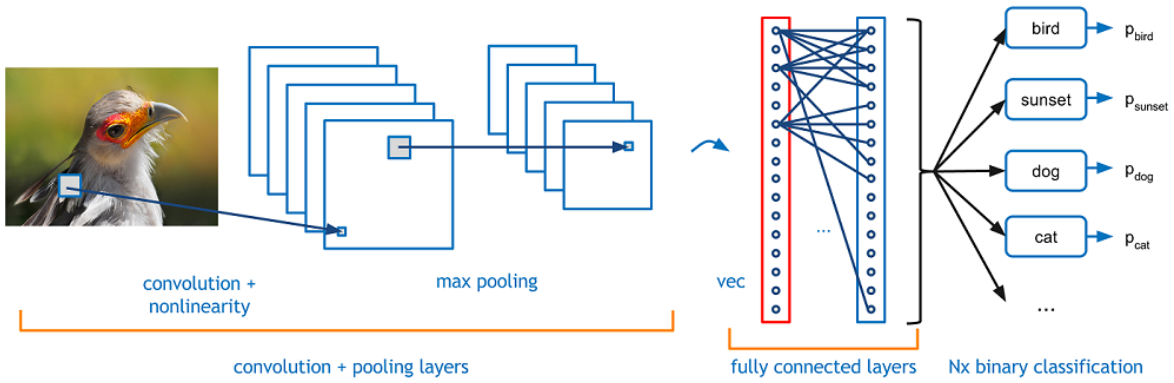

Figure 4 –This figure shows the different layer types of a Convolutional Neural Network (CNN). The input is image is fed through a series of convolutional and max pooling layers. After these, the model contains several fully connected layers, which are then connected to the output layer. In this image classification example, the neurons in the output layer give a probability for each class (p_{bird}, p_{sunset}, etc.), and the class with the highest probability is selected as the final output. Source: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

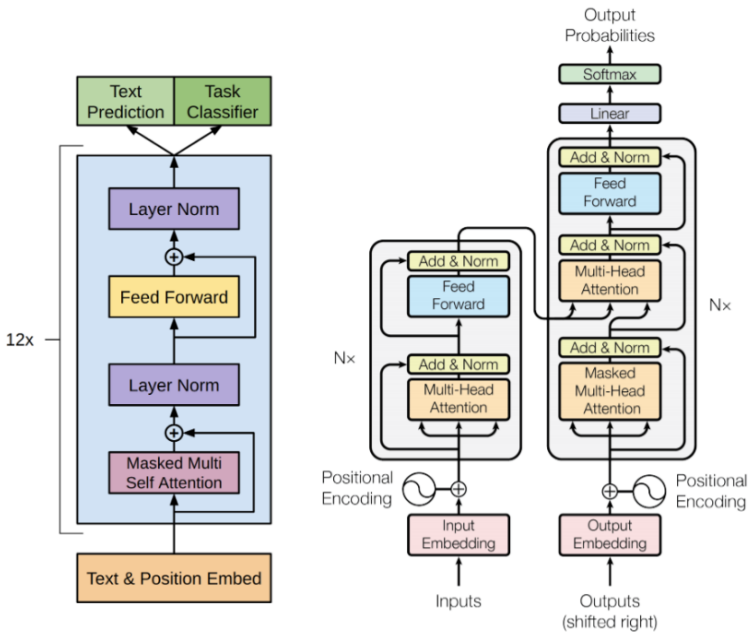

Figure 5 – This figure shows the architecture of the transformer block which is the core and most important part of any Large Language Model (LLM). Source: https://arxiv.org/pdf/1706.03762.pdf

Large Language Models (LLMs) (Fig. 5.) use a neural network architecture called a transformer. Transformers introduce self-attention mechanism, which allows the inputs to interact with each other and learn which other inputs they should pay attention to. This

mechanism is the key to the success of LLMs in natural language processing tasks, such as text generation, translation, content summary or sentiment analysis, where the meaning of a word can change based on its context within a sentence or document.

With the recent advancements in LLMs such as GPT4 or ChatGPT, there have been a growing interest in using these models in radiological reporting [28] [29].

LLMs provide a way to combine traditional free text reporting and structured reporting, by extracting the relevant information from a free text report based on a template. The template can serve as a prompt, which instructs the model to gather information from the text based on the template format. This allows radiologists to use the widespread way of dictation for reporting, but at the same time generate unified structured reports thus reducing problems arising from different styles of reporting. LLMs can also be used to propose different templates for a given disease or modality.

3. Aim and hypothesis

The primary aim of this thesis is to investigate the feasibility of web-based medical software solutions in general. More specifically, to build and test a fully-browser based neuroimaging software: we aim to develop WebMRI [30], an open source, cross-platform, web based neuroimaging platform that builds on top of the open source BrainBrowser. The goal is to combine the volumetric visualization capabilities of BrainBrowser with the image processing power of FSL, and create a web-based platform that can perform useful neuroimaging tasks, such as brain extraction and linear registration by running solely inside the browser, without using a web server or any browser plugins. Our goal is to port the FSL BET and FLIRT tools to WebAssembly, so that they can run in a browser environment, and be a part of WebMRI.

We propose another web based medical software, XReport [31]. With XReport our goal is not to push the performance boundaries of browsers, unlike with WebMRI, but to ship an easy-to-use, free and open source structured reporting platform for radiologist, with both template creation and reporting capabilities. We plan to compare it to other currently available template creation platforms.

With our web-based software systems we wish to advance browser based neuroimaging and structured reporting, and also promote open source medical software. We will to release our solutions on Github.

4. Materials and methods

4.1. Development of XReport, a web-based radiological structured reporting system

XReport software was written as a web application to support all operating systems and devices. Two programming languages were used throughout the development process, namely JavaScript and TypeScript. The project can be divided into two main parts: the library and the application. The library is a standalone module that implements the core features of the software: template building and reporting. The application can be any host that integrates the library, in our case it is a Single Page Application (SPA). Our workflow of creating reporting templates resembles intentional programming [32]. The programmer builds the foundation (template builder) of the software on top of which the domain expert (radiologist) can build the actual application (template). The programmer can later add custom dynamic logic to the template.

4.1.1. The library

The library exposes four public methods to interact with: *makeWidget*, *togglePreviewMode*, *getReportAsText* and *getTemplateForUpload*. The entry point is *makeWidget*. It can instantiate a new empty template builder or load a template from a Uniform Resource Locator (URL). Internally it creates an instance of each of the following classes: XReportDOM, XReportRender, Evaluator. The XReportDOM implements a custom subset of the Document Object Model (DOM) which allows only specific elements of the DOM or compositions of DOM elements to be used. The XReportRender calls the render methods of the XReportDOM entities and uses them to assemble either a builder or a viewer component, depending on whether the library is in editor or viewer mode. In editor mode the templates can be modified, whereas in viewer mode they are read-only, and are ready to generate reports. The Evaluator is an interpreter for our Domain Specific Language (DSL) called FormScript. It adds dynamic behavior to the templates through simple if-else logics and calculations. An example of a typical use case for FormScript is to show or hide a specific field if certain conditions are met, or to calculate a score for a scoring system. To view the

generated report the library exposes the *togglePreviewMode* method. Calling this method will transfer the viewer from reporting state to output state or vice versa. In output state the reporter can see the textual output of the form. The generated text can be accessed by the *getReportAsText* function. When the library is in editor mode a template can be saved by first getting it in JavaScript Object Notation (JSON) format with getTemplateForUpload and then sending it to a web service or storing it locally.

4.1.2. Template structure

The templates are composed of rows, which may have one or more groups in it. Groups are label-entity pairs, and entities are the form's input elements.

The JSON structure of a template is as following:

{ "formScript": "Form script source code is here", report: [{ XFormElem #1 }, { XFormElem #2 }…]}

General fields in XFormElem:

• type: defines what element to render, e.g. row, group, sel (select), mulsel (multiple select)
• id: a random generated unique identifier
• scriptAlias: an identifier/variable name by which FormScript can reference the field; auto-generated, but can be changed by user
• hideFromOutput: determines whether the value of the field should be visible in the generated text output
• hidden: determines whether the field should be rendered
• children: a list of groups in a row
• child: the entity of a group

There are fields specific to each entity but they are not listed here.

4.1.3. FormScript

FormScript is a DSL that is specifically designed to run inside XReport templates. It allows custom logic to be executed safely in forms, thus adding dynamic behavior to them. The script can be edited when the library is in editor mode and is accessible through the getScript library call. Once saved, it is stored in the same JSON file as the template itself.

The FormScript syntax is similar to that of JavaScript with some subtle syntactical differences.

Supported binary operations:

- addition (+)
- subtraction (-)
- division (/)
- multiplication (*)
- modulo (%)
- less than (<)
- greater than (>)
- less than or equal to (<=)
- greater than or equal to (>=)
- equal to (=)
- logical and (and)
- logical or (or)
- to the power of (^)

Unary operations:

- unary not (!)
- unary minus (-)
- unary plus (+)

Statements:

- expression
- assignment

- if
- function call

Types:

- string
- boolean
- number

Numerical and string literals are supported. The only variables that are allowed in FormScript are references to form elements. As mentioned earlier, variables are defined in the editor through the *scriptAlias* property. Function calls are defined only on variables. It is not allowed to declare functions neither are there predefined library functions without an element context. Calling a function has the following form: variable.function(…parameters). Functions should be defined for XFormElem classes. When XReport loads a report, it checks for an attached script. If there is a script attachment, it will start running it in an Evaluator instance.

4.1.4. Backend

Our SPA has a backend powered by Google Firebase to store the template resources. We store the template files in storage buckets. The metadata for each template, such as date of creation, creator's username, template name, template category is saved to Cloud Firestore documents.

The process of uploading a template to our backend includes the following steps:

• query template JSON from the library through getTemplateForUpload
• assemble upload metadata: date of creation, category, username, template name, template URL
• save the metadata to a Cloud Firestore document
• upload the template JSON file to the storage

4.1.5. Frontend

The frontend is built as a SPA using the Angular [33] and the Bootstrap Cascading Style Sheets (CSS) frameworks. Every icon used in the app are taken from the Font Awesome icon library. Angular supports client-side navigation, asynchronous data binding among others, which enables us to easily fetch and render views. To retrieve templates from Cloud Firestore we use the official Firebase JavaScript Software Development Kit (SDK) and the RxJS reactive programming library. In Firebase terms the templates form a collection, and individual entries in this collection are documents. To show these documents on the screen we followed the Model-View-ViewModel pattern with data binding.

4.1.6. Converting free text to structured report using LLMs

We experimented with automatic filling of our templates. We used GPT4 of OpenAI to generate structured reports from free text reports. The free text reports were test reports made up by us and did not contain real patient data in any form. We prompted the model with free text, and the template into which we want to convert the free text.
Here is an example prompt snippet to instruct the model to convert a rectum primer tumor staging report into the corresponding template format available on the XReport demo website:

We will provide a free text radiological report, and a structured radiological report template. Convert the free text format to the template format.

Free text:
A polypoid, mucinous mass is visible 3mm from the mesorectal fascia. The mass infiltrates the external sphincter. The length of the mass is 50mm. There are 5 enlarged lymph nodes, 2 of them less then 5 mm in size, and 3 of them larger than 9mm. There are 2 tumor deposits inside the mesorectal fascia.

Template:

{

„morphology”: [„polypoid”, „ulcerating”, „circumferential”, „semicircumferential”],

„selectedMorphology”: „”,

„appearance”: [„mucinous”, „non-mucinous”], „selectedAppearance”: „”,

„lengthOfTumor”: { „value”: „”, „unit”: „mm” },

„sphincterInvolvement”: [„none”, „internal sphincter is involced”, „intersphincteric space is involved”, „external sphincter is involved”], „selectedSphincterInvolvement”: „”,

„distanceTumorMesorectalFascia”: { „value”: „”, „unit”: „mm” }

„lymphNodeInvolvemnet”: False,

„numEnlargedLymphNodes”: 0,

„numLymphNodesLessThan5mm”: 0,

„numLymphNodesGreaterThan9mm”: 0,

„tumorDepositInsideMesorectalFascia”: False,

„numTumorDeposites: 0

}

The structure of the prompt gives hint to the model to what the task is. We guide the information gathering with a template, so we call this method template-guided LLM-based structured reporting. The guiding is both explicit and implicit: in some cases the property naming is explicit about what kind of information should be extracted there, such as „numLymphNodesLessThan5mm”. However, in other cases, such as „selectedMorphology”, the naming implicitly contains the information that whatever description the model finds about tumor morphology, it should chose from the „morphology” list. The template can be in any textual format, we chose JSON because the XReport templates are JSON files. The model output and the interpretation of the output will be discussed in the results section.

4.2. Development of WebMRI, a web-based modular neuroimaging platform

The development of WebMRI consists of three main parts:
- the porting of the brain extraction (BET) and linear registration (FLIRT) FSL tools to WebAssembly
- extending BrainBrowser with a plugin system, and integrating the ported tools into BrainBrowser using this system
- creating a demo application to showcase multiple neuroimaging workflows using the ported tools

4.2.1. Porting FSL BET and FLIRT to WebAssembly

For the porting we had to study the source code of FSL. It is essentially a modularly built software system, where each component can be separately compiled and run as a command-line application. BET and FLIRT are such components of the FSL.

The Emscripten porting (transferring from C++ to WebAssembly) consisted of the following sub-processes:

1. Replacing the default compiler (gcc on Linux) with the Emscripten Compiler Frontend (emcc).
2. Compiling all libraries required for the translation of BET and FLIRT (meshclass, newimage, prob, miscmaths, fslio, niftiio, znz, newmat, utils, zlib).
3. Correcting errors that occurred during compilation, then recompiling.
4. Compiling BET and FLIRT to Low Level Virtual Machine (LLVM) bitcode.
5. Compiling the LLVM bitcode to WebAssembly.
6. Writing a web worker that runs the program separate from the user thread, so that the user interface is responding while the operation is still ongoing.
7. Creating a user interface through which you can communicate with the web worker.

The porting is based on the versions FSL 3.3.11, BET 2.1, FLIRT 5.4.2. We named the ported FSL modules bet2.js and flirt.js.

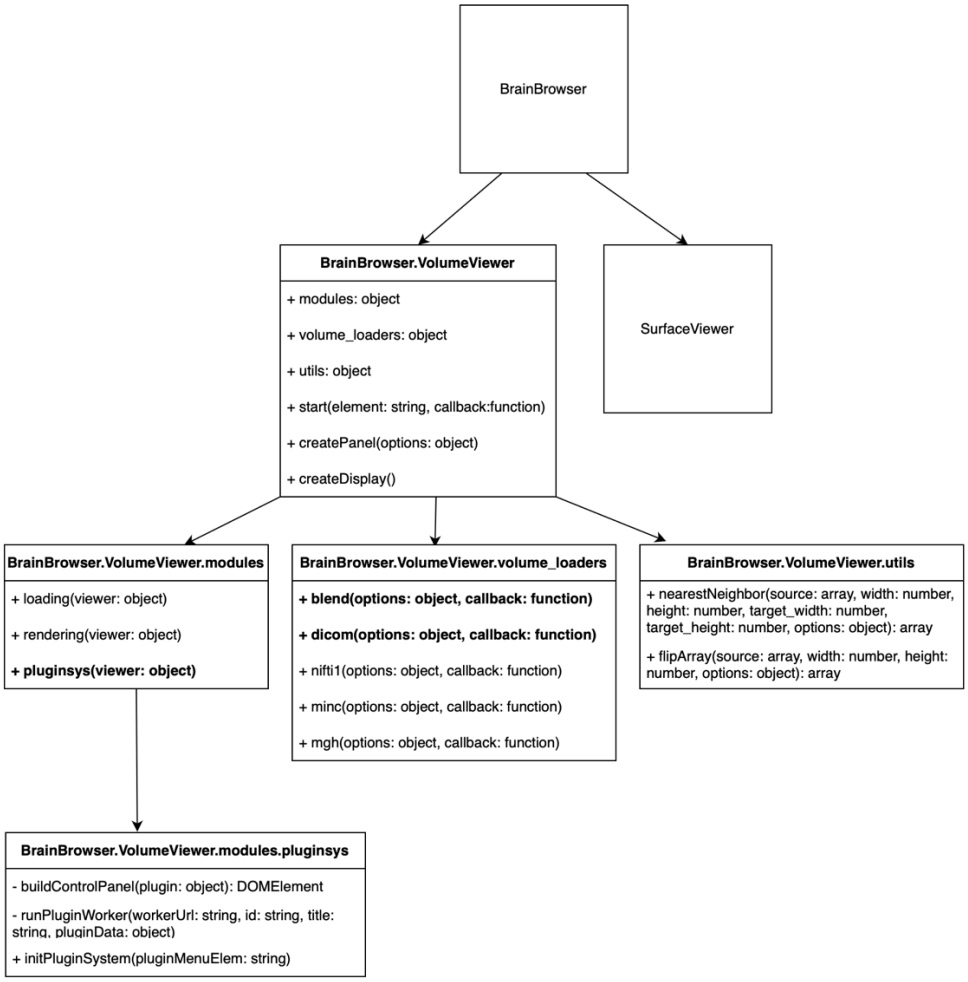

Figure 6 – The modified architecture of BrainBrowser: we extended BrainBrowser with a plugin system, a dicom and a blend volume loader.

4.2.2. The DICOM volume loader

By default, BrainBrowser supports loading files in NIfTI, Medical Imaging NetCDF (MINC), and Massachusetts General Hospital (MGH) formats, but we extended it with two additional volume types: "dicom" and "blend".

In the field of neurological image processing, the gold standard format is NIfTI. However, to make our software potentially usable in a clinical environment in the future, it was necessary

to support the raw DICOM format as well, as the recordings are directly accessible from the MR scanners in this format without extra conversion.

In our software, loading DICOM volumes involves an implicit conversion step. If dicom is present as the volume type in the input parameter of the *viewer.loadVolume* call, then the selected files are first converted to NIfTI by the dicom volume loader, and then the resulting volume is loaded through the nifti1 loader. Conversion is necessary because all post-processing and other algorithms in the software already work on the NIfTI format. This volume loader is the first component of our software system to be successfully converted from C++ to WebAssembly using Emscripten and integrated into our architecture. The core of its operation can be summarized in the following code snippet:

```
var worker = new Worker("dicom2nifti-worker.js");
worker.addEventListener("message", function(e) {
var niiFromDicom = e.data;
VolumeViewer.volume_loaders["nifti1"]({ type: "nifti1",
nii_raw: niiFromDicom
}, callback);
});
worker.postMessage(result_files);
```

The first line creates a web worker for the conversion algorithm, which allows it to run separately from the user thread, so the program responds to user interactions throughout the conversion. We pass the files to be converted to the worker through the *postMessage* function. The second line creates an event listener on the worker, which listens for the "message" event. When the conversion is finished, the worker passes the completed NIfTI volume to the event listener, which is then loaded (last line of code). The program performing the conversion is originally a command-line C++ program. The advantage of Emscripten porting is that the WebAssembly version of the program also accepts all command line parameters:

```
var Dicom2NiftiModule = {
...
arguments: ["-z", "n", "-f", "%p_%t_%s", "-o", "/niiOut", "/dicomIn"],
```

...
};

4.2.3. Plugin system

To allow running our ported tools in BrainBrowser, we extended it with a plugin system (Fig. 6.). The plugin system is defined as a VolumeViewer module. It is responsible for initializing and running the plugins registered in the program.
We register the available plugins through the config module of BrainBrowser.

```
BrainBrowser.config.set("plugins", [{
    name: "Brain extraction",
    title: "Brain extraction",
    author: "https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FSL",
    id: "bet",
    worker: "src/brainbrowser/volume-viewer/workers/bet-worker.js",
    gui: "plugin-GUIs/bet-menu.json"
  }
 ]);
```

Plugins are defined as an array of plugin objects. Each object describes a specific plugin with "name", "title", "author", "id", "worker," and "gui" attributes. The "author" property stores a link to the website of the author of the plugin, in our case, FSL. The "id" is a unique identifier for the plugin. The "worker" is a link to the web worker that invokes the WebAssembly module, and the "gui" is a link to the JSON file that describes the user interface of the plugin, which shows tweakable parameters and a button to run the tool.
What to render on the Graphical User Interface (GUI) and what action to perform when the plugin is executed is defined by a JSON, as shown below.

[{"type": "file", "text": "Input volume"}, {"name":"-f", "type": "number", "min": 0, "def": 0.5, "max": 1, "text": "Fractional intensity threshold"}, {"name": "-o", "type": "bool", "text": "Brain outline mask"}...]

When the user clicks on the "Run" button on the plugin dialog, the UI handler loops through the input fields and creates name-value pairs, which in turn are passed to the web worker. The web worker will then run the WebAssembly program with the received command-line arguments.

5. Results

5.1. XReport

5.1.1. Template viewing and building

The viewer/builder page is where we load our templates and render them with our library as show in Fig. 7. The form is centered horizontally and have a slight drop shadow around it. There is a button group on the right side of the form which contains different buttons based on which state the page is currently in (viewer or builder).

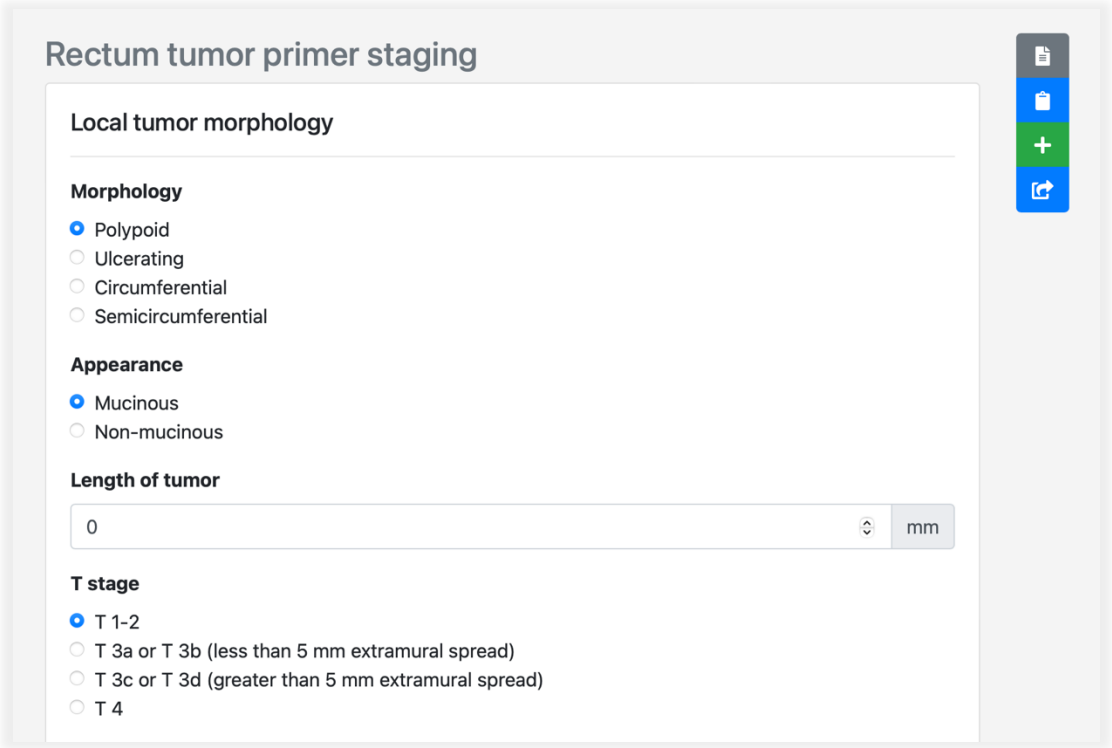

Figure 7 – A rectum tumor primer staging template rendered in XReport. There are various elements to build the template from, such as single choice, numerical or multiple choice fields. More complex use-cases, such as scoring systems, can also be created leveraging advance components such as a rating table, in combination with dynamic behaviour added to the form using our own DSL FormScript.

Buttons in viewer state:

- preview report: shows the textual output of the report in its current state
- copy to clipboard: copies the textual output onto the clipboard so that the report can easily be copy-pasted into other systems without tight integration
- new report: discards the current state of the form and opens it again
- share: copy the template link to clipboard so that it can be shared with colleagues

Buttons in builder state:

- save template
- discard template

There are 13 components to choose from when building a template:

- Text field: It is an input field which allows users to enter text. It is not resizable.
- Plain text: It is a read-only user interface element that shows multiline text.
- Number field: It is an input field which only allows numeric entries.
- Calculated field: It is a read-only number field used to show computation results.
- Boolean field: It is a checkbox that supports boolean entries: the state is either true (checked) or false (unchecked).
- Single choice: It is a component that shows a list of possible choices, and allows only one to be selected.
- Multiple choice: It is a component that shows a list of possible choices, and allows any combination of them to be selected.
- Textarea: It is a resizable text field.
- Date: It is a component that allows selecting a date from a calendar.
- Header: It is a read-only textual component that can be used for section heading.
- Information: It is a component similar to plain text, but it has a colored background to draw the attention of the user. It is used to give hints to a given field or a section of the template. It is hidden behind a question mark icon: if the text is not visible and the icon is clicked, the text will be visible or vice versa.

- Rating scale: It is a table which renders a checkbox in each of its cells. In every row there can only be one cell selected, so it can be thought of as a variation of the single choice component.
- Image: It shows an image based on the URL that was given by the template creator.

If we take an oncological example, a question regarding the size of the tumor would be a number field, a Tumor, Node, Metastasis (TNM) staging system could be created using single choice fields, or a rating scale, etc. Every component is added to the form with a label attached. A component-label combination is called a group. Groups are added to rows, and rows are added to sections. The sections make up the whole template.

5.1.2. Component editors

Every component has an editor view as shown in Fig. 8. Every component type has its own editable properties. For example, the input field has a unit property (mm, cm, etc.), a single choice field has an options property, an image has an URL property. The component editor is activated by hovering the mouse over the component, then clicking on the pencil icon. Components can be deleted by clicking on the minus sign.

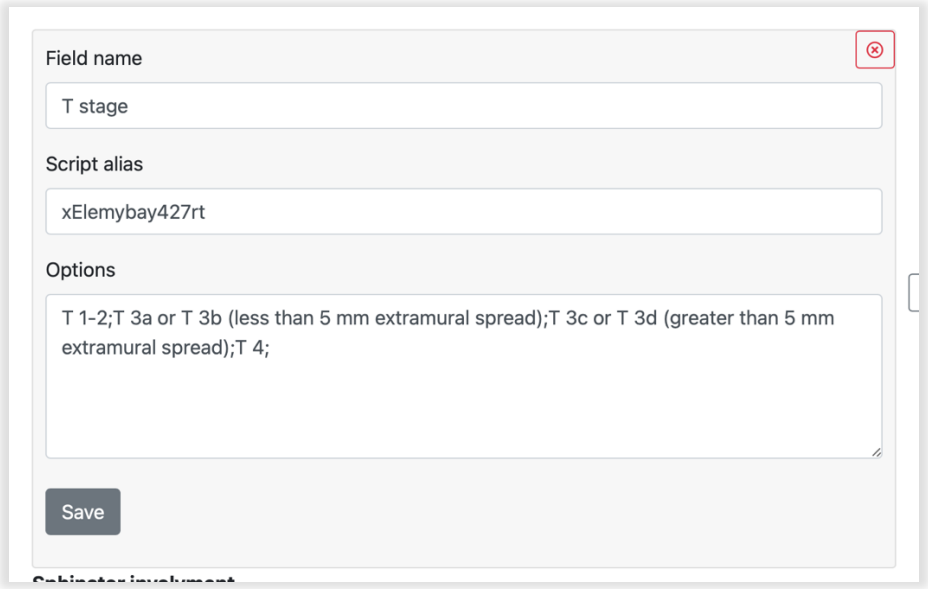

Figure 8 – Editor view: every component has a predefined editor view, which allows modifications of the given field.

5.1.3. Row editors

Row operations can be performed by clicking on the three vertical dots at the end of each row. The click event will trigger a secondary menu to open with all the components, and two actions: delete and duplicate.

5.1.4. FormScript editor

On the main builder component there is a button with a branch icon which toggles the view between template editing and FormScript editing. The FormScript editor as shown in Fig. 9. is a simple resizable text area where the user can edit the dynamic logic that is attached to the template. When switching back from script editing, the script is automatically evaluated and the changes are visible.

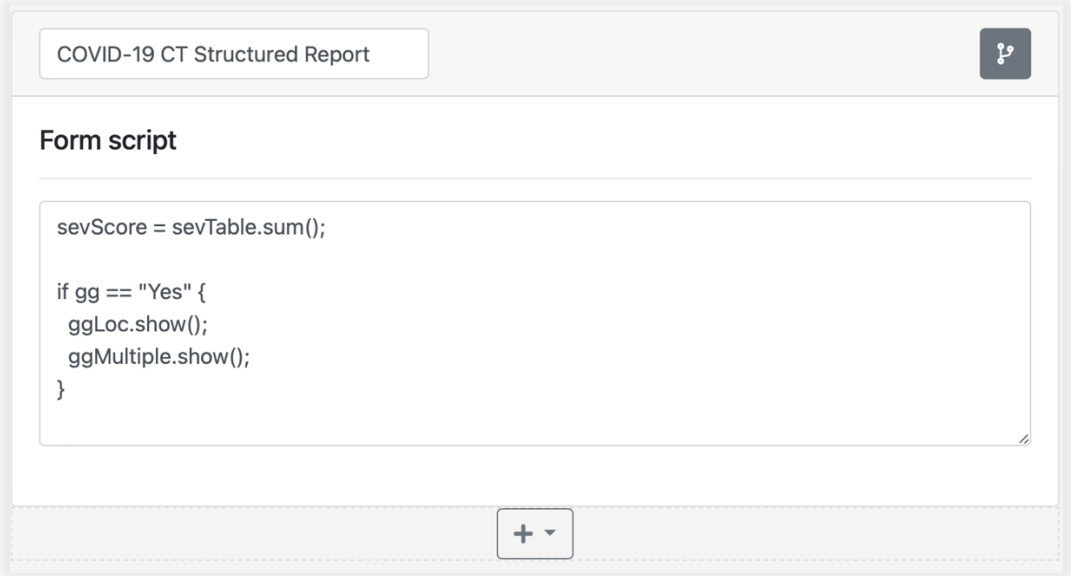

Figure 9 – FormScript: Code snippets can be attached to templates to give them dynamic behavior. It is a simple DSL similar to JavaScript, but very limited compared to it. The main reason it was developed is safety.

5.1.5. Converting free text to structured report using LLMs

For the prompt presented in section 3.1.6, GPT4 generated the following output:


```
{
"morphology": ["polypoid", "ulcerating", "circumferential", "semicircumferential"],
"selectedMorphology": "polypoid",
"appearance": ["mucinous", "non-mucinous"], "selectedAppearance": "mucinous",
"lengthOfTumor": { "value": 50, "unit": "mm" },
"sphincterInvolvement": ["none", "internal sphincter is involced", "intersphincteric space is involved", "external sphincter is involved"], "selectedSphincterInvolvement": "external sphincter is involved",
"distanceTumorMesorectalFascia": { "value": 3, "unit": "mm" },
"lymphNodeInvolvement": true,
"numEnlargedLymphNodes": 5,
"numLymphNodesLessThan5mm": 2,
"numLymphNodesGreaterThan9mm": 3,
"tumorDepositInsideMesorectalFascia": true,
"numTumorDeposites": 2
}
```


Using only a single prompt, and without any fine-tuning of the model, it understands the JSON structure we would like to get as an output. It correctly maps the information available in the free text to the relevant property in the JSON, even if a given parameter has to mapped to a nested property, such as in the case of length values (lengthOfTumor), where there is a „value” and a „unit” nested property.

We modified the original template to test if the model can gather information that was not explicitly mentioned in the free text report. After leaving out mentioning the total number of enlarged lymph nodes, the model generated the following answer:

```
The total number of enlarged lymph nodes was not explicitly mentioned in this free text. Given that 2 of them are less than 5mm and 3 are greater than 9mm, we can
```

infer that there are at least 5 enlarged lymph nodes in total. Here's the converted template:

Then it proceeds to present the same JSON as above. It correctly infered the total number of lymph nodes.
Changing the units of the length measurements also work. By replacing the sentence „The length of the mass is 50mm” to „The length of the mass is 5cm”, GPT4 generates the following answer:

The length of the tumor is given in centimeters in the free text report, so we'll need to convert this to millimeters to match the unit in the template.

After that it generates the correct JSON. This shows how capable LLMs are in analyzing radiological reports. This LLM based approach has the potential to fill the gap between traditional dictation based and template based reporting: instead of thinking in templates, and manually filling out a form, the doctor can report the case as usual, and then copy-paste the report to an LLM based system where the free text and the selected template will be fed to the LLM as a prompt, and based on the generated structured format the form can be filled automatically, and the structured data can be saved to a database for further processing. This brings together the benefits of both worlds: the speed of dictation, and the quality of structuring.

5.2. WebMRI

The entry point of our demo application showcasing WebMRI is the dashboard (Fig. 10.). There is a navigation bar on the top with menu items "File", "Tools", and "About". The "File" menu hosts actions such as opening DICOM series, a NIfTI volume, or creating a blended volume ("Create overlay") out of two input volumes. The "Create overlay" functionality is an important last step in a linear registration workflow, as it simultaneously visualizes the two volumes registered to each other. The "Tools" menu shows the list of the plugins available in the application (BET and FLIRT), and through the "About" menu, users can access the developer documentation and user manual of WebMRI.

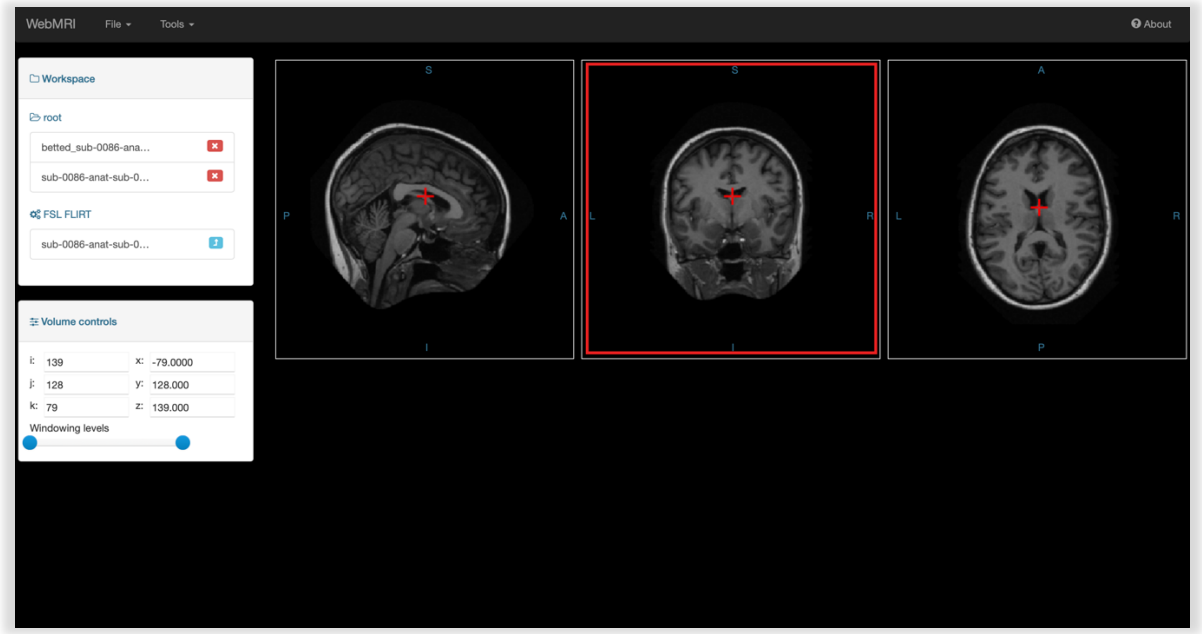

Figure 10 – The image displays the user interface of WebMRI. At the top, the menu bar is shown. Below the menu bar on the left is a window labeled "Workspace", which displays the files loaded and those generated by the plugins. Below that, under "Volume controls", elements that allow for manipulation and navigation of the loaded image are visible. On the right, the loaded volume is shown in sagittal, coronal, and axial sections.

Under the navigation bar, we present the main screen, which is divided into the rendering section on the right, and the panel section on the left. The rendering section hosts the BrainBrowser VolumeViewer widget, which visualizes the loaded volume in sagittal, axial, and coronal planes. The panel section next to it hosts the "Workspace" and "Volume controls" widgets.

The "Workspace" widget (Fig. 11.) shows the files currently opened in the application. These files can either be loaded by the user (input files) or generated by plugins (output files). The workspace has a root folder, which is a collection of files that are visible to plugins as input files and plugin-specific folders, into which plugins generate their output files. If the users want to further process output files, they first have to move the relevant files to the root folder by clicking on the blue up arrow next to the file names. This way, the selected files will be visible to the plugins. The "Volume controls" widget contains coordinate information, blending controls, and windowing sliders.

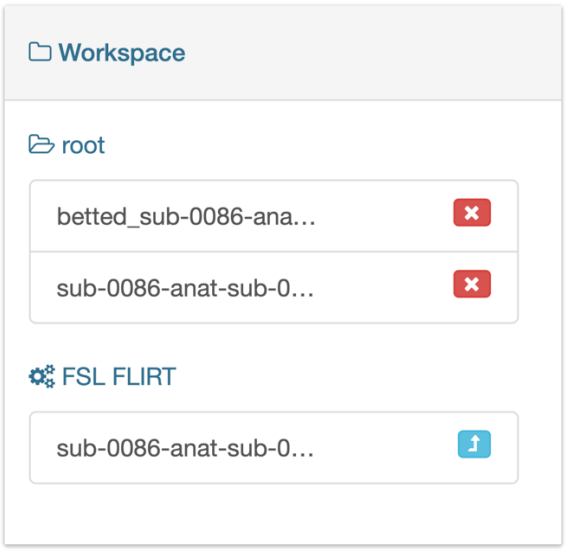

Figure 11 – Workspace widget: The workspace has a simple folder structure where the input files are listed under the „root" folder, and plugins generate their outputs into plugin specific folders, such as FSL FLIRT in this case.

Our demo application showcases our two ported FSL tools. They can be accessed from the "Tools" menu. Fig. 12. shows the automatically generated user interface for BET, and figure 13. illustrates the GUI for FLIRT. The input volumes can be selected from a drop-down menu. The user is presented with a list of modifiable parameters, which will translate to command-line arguments by the plugin system when the "Run" button is clicked, and the underlying tool is invoked.

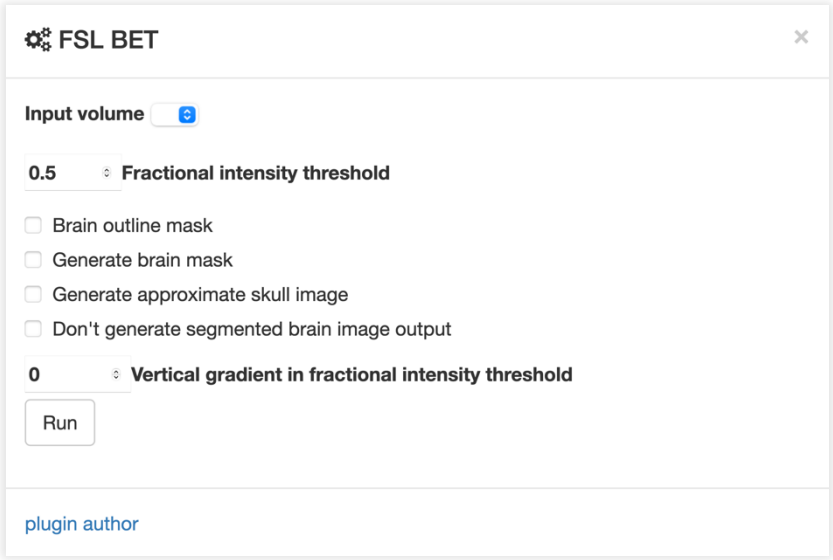

Figure 12 – Automatically generated user interface for the ported FSL BET tool.

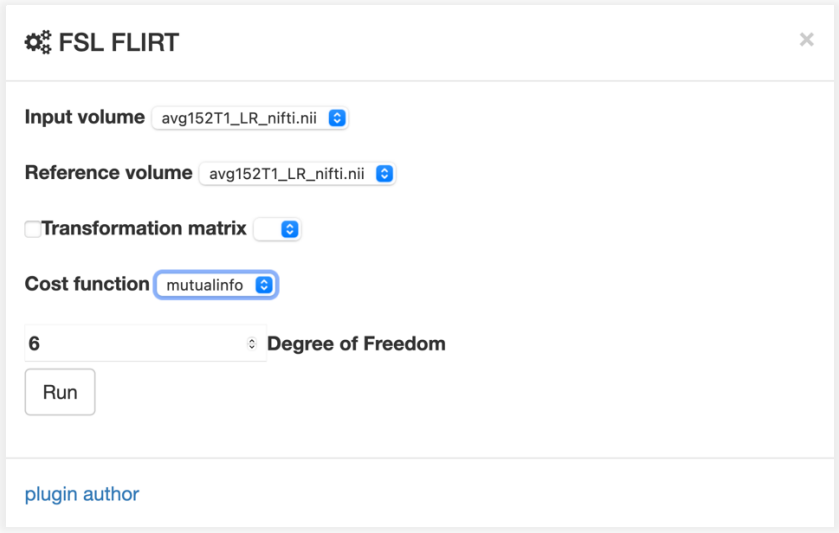

Figure 13 – Automatically generated user interface for the ported FSL FLIRT tool.

The most practical use case our system is capable of supporting in its current form is the multi-modal assessment of neuroimaging data. As an example, let us look at how cerebral microbleeds can be detected in an MRI volume. The most widespread way of analyzing microbleeds is to leverage susceptibility weighted imaging (SWI). Although SWI can help identify microbleeds with high sensitivity, their localization is challenging relying solely on this modality. T1-weighted MRI images preserve the anatomical features of the brain, so using the T1 modality in combination with SWI can help with localization. WebMRI can co-register an SWI to a T1 MRI volume thus enabling precise microbleed detection. To enhance registration result brain extraction can also be performed. Since our system can handle DICOM to NIfTI conversion, the only step the user has to perform before loading data into our system is to export the DICOM series from the PACS.

5.2.1. Performance evaluation of the ported FSL tools

We compared bet2.js and flirt.js with the original C++ (native) versions in terms of performance. Both the native and asm.js versions were compiled at the -O3 (highest) optimization level. The native versions were run on the Windows 10 Linux subsystem, and the two WebAssembly versions were run in Google Chrome, then in Mozilla Firefox browsers. The configuration of the computer used for testing is as follows:

- CPU: Intel® Core™ i5-3230 @ 2.60Ghz
- RAM: 6.00 GB
- System type: 64-bit operating system, x64-based processor

We ran bet2.js and the native BET on a total of six test volumes. The performance testing of the flirt.js and the FLIRT C++ version was carried out using ten test volumes, which formed five pairs. From each pair of volumes, one was the input volume and the other was the reference volume to which the former was registered. We chose SWI recordings for the input volumes and T1 recordings for the reference volumes. In case of flirt.js, the SWI-T1 volume pairs come from the same subject, and the volumes were brain extracted as a preprocessing step. All volumes were in NIfTI format. The runtimes measured during the tests were finally depicted on bar charts (Fig. 14) (Fig. 15).

5.2.1.1. Performance evaluation of bet2.js

In the case of the native BET version, the average runtime was 2.96 seconds. For bet2.js, when using Google Chrome, the average runtime was 5.75 seconds, while on Mozilla Firefox, it was 4.62 seconds. Therefore, the WebAssembly program ran on average 1.94 times slower than the native version with Chrome, and only 1.56 times slower with Firefox. During testing with Firefox, we achieved a 25% performance increase compared to Chrome (performance calculated from the reciprocal of the runtime) (Fig. 14).

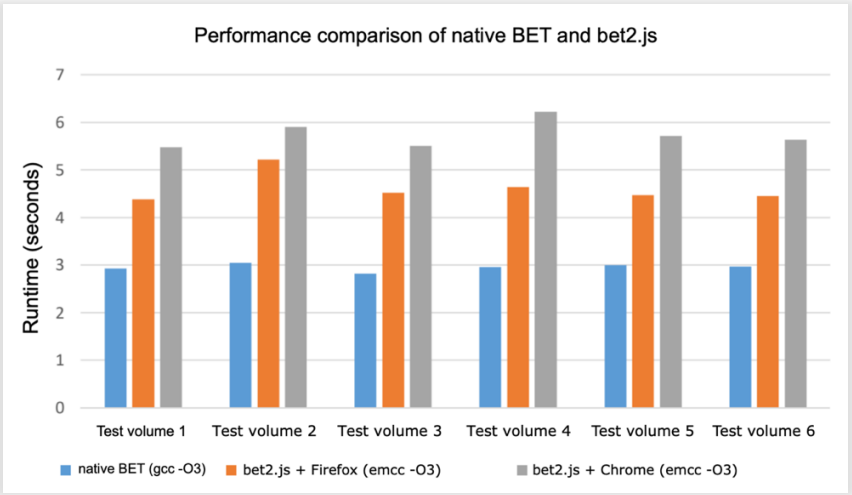

Figure 14 – Comparison of the runtime of the native BET and the bet2.js programs. The vertical axis shows the runtime expressed in seconds. The 6 column groups represent the processing times for the 6 test volumes: the blue column represents the native, the orange represents the time measured in Firefox, and the gray represents the runtime in Google Chrome.

5.2.1.2. Performance evaluation of flirt.js

The native FLIRT completed the tests in an average time of 22.12 seconds. The WebAssembly version, when run on Firefox, executed the tasks with an average runtime of 40.79 seconds, and on Chrome, it took 47.64 seconds. Compared to the native version, the WebAssembly program ran on average 2.15 times slower on Chrome. On Firefox, the slowdown was 1.84 times. In this test too, Firefox outperformed Chrome, this time, however, by only 17%. The details of the comparison can be seen in the diagram below (Fig. 15).

Regarding memory usage and stability we found that in highly memory constrained scenarios where there are a lot of browser processes running at the same time (a lot of tabs open at the same time), the ported tools may behave unpredictibly and can crash due to the tab in which the FSL programs are hosted run out of memory. Identifying these memory constrained scenarios is challenging, as they are dependent on many factors.

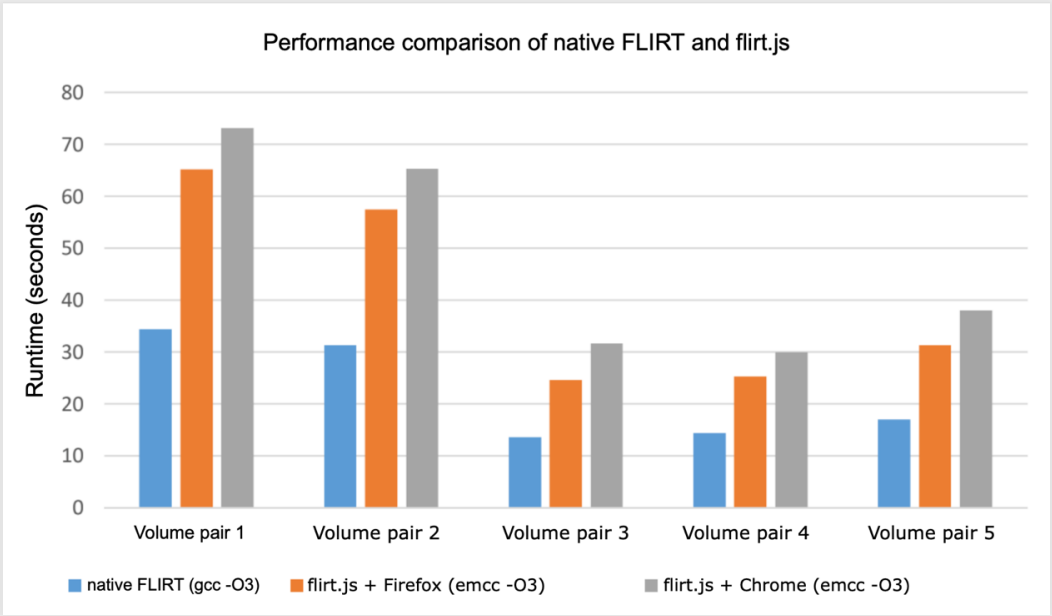

Figure 15 – Comparison of the runtime of the native FLIRT and the flirt.js programs. The vertical axis displays the runtime expressed in seconds. The 5 column groups represent the processing times for the 5 volume pairs: the blue column represents the native, the orange represents the time measured in Firefox, and the gray represents the runtime in Google Chrome.

6. Discussion

We developed browser-based medical software systems in two fields, namely radiological reporting and neurological image processing. Although XReport and WebMRI are two different systems with no direct interoperability, they have a lot in common. First and foremost both are open source systems which are released on Github for other developers to use and modify. Both of the proposed software features modular design: XReport can be integrated into other applications easily, and the supported template modules can be extended, and WebMRI allows third-party developers to write plugins to it. They both were developed to pave the way for free, open source, browser-based medical software systems. By running them client-side in a web browser, they are cross-platform and require no installation. This reduces setup and maintanance costs, and requires no special effort to keep them up-to-date, as new versions are immedietaly available. Besides promoting and advancing open source medical software, we had specific goals with the two systems. With XReport we aimed to build an easy-to-use structured reporting template creation and reporting platform to facilitate template creation, and advance templated radiology. With this software we did not aim to demonstrate high-tech browser capabilites, unlike with WebMRI. The main goal with WebMRI was to investigate the feasibility of fully browser-based neuroimaging by leveraging advanced web technologies such as WebAssembly or WebGL. We managed to port parts of the most popular neuroimaging framework FSL to WebAssembly, and run it in a browser without special plugins. By porting BET and FLIRT to the web, ours is the first fully browser-based system that is capable of brain extraction and linear registration without using a web server. These features enable performing multimodal assessment of neuroradiological data inside a web browser. Although the native algorithms are faster, we found that in everyday use, our ported versions are not significantly than the native ones.

In the following sections let us discuss our two open source software systems separately.

6.1. XReport

With XReport we built a free, cross-platform structured reporting platform for radiologists. It enables both creating and viewing reporting templates in an easy, user-friendly way. We built our software with modular design in mind and refactored the core features into a separate library to make embedding it into other products easy. We also built an application with the library embedded in it to demonstrate the easy integration. Furthermore, we designed a simple DSL called FormScript to add dynamic logic to our forms. The main feature of it, and the reason we created it in the first place, is security. It does not allow malicious code executions unlike the eval function of JavaScript. It is also very simple to use because of its limited feature set. Our templates are dynamic, responsive and have modern design. The templates generate easy to copy–paste structured textual output to be compatible with any HIS, and to integrate well into dictation-based workflows. Our templates help not only in precise reporting, but also serve as a guide for radiologists thanks to our custom form elements such as images and rating tables.

We compared our solution to a similar free service developed by RSNA. From a technological point of view both programs are similar since they are built using web technologies but they have their differences when it comes to the ecosystem, editing process and user experience. The RSNA template library has a more mature ecosystem: there are a lot of contributors who build and upload templates, there are some nice to have features such as favouriting a template. But the template editing itself is less advanced than ours. In the RSNA editor the screen flow to get to the actual editing is as following: click on “Create and Upload a Template button”, click on “T-Rex Template Editor”, interact with a popup which asks how the user wants to start the editing, click on one of the options. In our program the screen flow is a lot simpler: click on “Add new template”, and you are in the editor. In the RSNA editor adding individual elements has some issues. The elements have to be drag and dropped from a side panel, which is problematic on mobile devices as there is not enough space. The element editor works as a pop-up which brings the user out of the editing context. In our app adding elements is responsive (works on mobile devices as well), and is inline, so the user remains in the editing context throughout the whole process. When it comes to how dynamic the templates are we found that RSNA templates do not allow dynamic behavior such as hiding/showing elements based on certain conditions. Through FormScript our system enables fully dynamic behavior. The RSNA editor lacks some important elements such as

images and rating tables which are essential in information sharing and oncological grading systems.

We experimented with automatic free text to structured report conversion using the GPT4 LLM. Our preliminary findings indicate that even without fine-tuning, GPT4 can deliver precise results when guided by a template. This promising observation paves the way for further enhancement of the system's accuracy by specifically fine-tuning an LLM on radiological reports. Given the large array of radiological pathologies and their textual representations, this specificity could lead to significant improvements in results.

Beyond just template-guided information extraction, LLMs can be prompted to select the most appropriate reporting template for a given case. This additional feature offers more flexibility in the reporting process, as it eliminates the need for pre-loading a template. In other words, the LLM can take into account the context provided in the report to select and adhere to the best-suited template.

Moreover, introducing small alterations to the prompts, such as designating a certain property as "required", can dramatically improve the functionality of the system. For instance, this modification can enable the LLM to flag missing key information in a report, thereby enhancing the safety features of the system. This approach effectively shifts the focus from a strictly structured template-based approach to a more adaptable, flexible, and safety-conscious reporting model, which can have profound implications for the field of radiology.

6.2. WebMRI

With WebMRI, we built a cross-platform, extensible neuroimaging platform that supports brain extraction and linear registration and can be run in any modern web browser without using third-party browser plugins or a dedicated web server. For brain extraction, we used the BET, and for linear registration, the FLIRT tool of the popular neuroimaging library collection FSL. For the porting process, we used Emscripten to convert the native programs to WebAssembly binaries. We extended BrainBrowser with a plugin system to be able to load our ported tools into the system combining volumetric visualization with image processing. We built a demo application that incorporates the ported tools and demonstrates that a complete neuroimaging workflow from DICOM loading to linear registration can be performed inside a web browser.

Building on the foundations of BrainBrowser, we increased the scope of browser-based neuroimaging and showed that not only neuroimaging data visualization but computationally intensive processing algorithms could also be performed inside modern web browsers. In contrast to Slicer 3D (to which our system is similar in terms of modularity) or other non-browser-based programs, our solution does not require an installation. By removing the need for a web server, and complex setup processes, we reduced the cost and improved the ease of use of certain neuroimaging workflows. These factors can improve the clinical adoption of new image- processing tools. Although we demonstrated the porting of two popular neuroimaging tools, there is no limit to bringing other algorithms into the WebMRI plugin system as long as they are written in a programming language that can be compiled to WebAssembly. As more and more programming languages used in computationally intensive application development (such as Rust) support compilation to WebAssembly, the number of libraries that can be ported to WebMRI will increase.

The limitation of our fully browser-based system is twofold. The memory usage of heavy workloads, such as processing large numbers of volumes in bulk, might exceed the resources available inside a browser tab. There is also a performance penalty for the porting due to the overhead of the WebAssembly runtime compared to fully native execution. The impact of these limitations varies between programs and browsers and can be mitigated using optimizations in the Emscripten toolchain or the original codebase. If these factors make it not possible to reliably run a given tool in the browser, it can still be run in a web server since Emscripten supports execution in a Node.js runtime.

In the future, we plan to increase our fleet of ported neuroimaging tools and make our plugin system more modular and extensible. We aim to make it easier to integrate our system into a clinical PACS environment, so there will be no need to export DICOM files. We also plan to support hybrid execution so that a given algorithm can be run on the client or server side, depending on the requirements of the given workload. We plan to improve the visualization capabilities of WebMRI by adding support for customizable volume viewer widgets and annotation of points of interest on the volumes.

7. Summary of the new scientific results

1, We developed WebMRI, a web-based, modular neuroimaging platform. We ported the FSL BET and FLIRT (brain extraction and linear registration) image processing algorithms to WebAssembly so that they can be run in a browser environment. To the best of our knowledge, no other web port of these FSL tools exist. We compared the runtimes of the native and ported FSL tools, and found that in everyday use, our versions are not significantly slower than the native programs. We added support for DICOM loading in WebMRI, thus eliminating the need for an external DICOM to NIfTI conversion step. We developed a plugin system, which allows other developers to create new algorithms, or port existing ones, and bring them into the WebMRI platform.

2, We developed XReport, a free and open-source, web-based structured reporting platform for radiologists, which supports both template creation and reporting in a user-friendly manner. We developed an LLM-based solution for automatic structured reporting template filling from free text report, using prompt-engineering techniques.

8. Acknowledgements

I would like to express my sincere gratitude and thanks to my program leader Prof. Dr. Bogner Péter for his support and guidance throughout the years leading up to this PhD thesis.

I am greatly thankful to my supervisor Dr. Arnold Tóth for his support and technical excellence. He helped conceptualize WebMRI, and thanks to his experience in using the FSL tools he gave me valuable feedback and directions during the development and testing of our software.

I am grateful to Dr. Ferenc Kövér for his involvement in the XReport development. He helped popularize XReport among his colleagues at the Diagnostic Center of Pécs, and also created multiple templates in the program. His feedback and guidance was really valuable to shape and fine-tune XReport.

I am thankful to Dr. Sándor Szukits for his involvement in testing the template creation and reporting workflow of XReport. His experience as a user with other structured reporting platforms was beneficial in building our software.

I am thankful to all the radiologists at the PTE Department of Medical Imaging, and the Diagnostic Center of Pécs, who have been using XReport. Their feedback helps make XReport better and more reliable.

At last but not least I would like to thank my wife Margaréta and my sons Ádám, Sebestyén and Bendegúz, for their love and support throughout these challenging years. I am grateful to my whole family who encouraged me to pursuit this research, and helped me through the hard times.

9. References

[1] M. Mitchell and L. Kan, “Digital Technology and The Future of Health Systems,” *Health Systems & Reform*, vol. 5, Feb. 2019, doi: 10.1080/23288604.2019.1583040.

[2] P. Lambin *et al.*, “Radiomics: the bridge between medical imaging and personalized medicine,” *Nat Rev Clin Oncol*, vol. 14, no. 12, pp. 749–762, Dec. 2017, doi: 10.1038/nrclinonc.2017.141.

[3] C. Dong *et al.*, “Characterizing browser-based medical imaging AI with serverless edge computing: towards addressing clinical data security constraints,” *Proc SPIE Int Soc Opt Eng*, vol. 12469, p. 1246907, Feb. 2023, doi: 10.1117/12.2653626.

[4] R. L. Bashshur, E. A. Krupinski, J. H. Thrall, and N. Bashshur, “The Empirical Foundations of Teleradiology and Related Applications: A Review of the Evidence,” *Telemed J E Health*, vol. 22, no. 11, pp. 868–898, Nov. 2016, doi: 10.1089/tmj.2016.0149.

[5] G. E. Moore, “Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.,” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, Sep. 2006, doi: 10.1109/N-SSC.2006.4785860.

[6] “Cornerstone.js | Cornerstone.js.” https://cornerstonejs.org/ (accessed Aug. 29, 2023).

[7] T. Sherif, N. Kassis, M.-É. Rousseau, R. Adalat, and A. C. Evans, “BrainBrowser: distributed, web-based neurological data visualization,” *Frontiers in Neuroinformatics*, vol. 8, 2015, Accessed: Aug. 29, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fninf.2014.00089

[8] R. C. Verma, H. M. Wu, A. J. Duerinckx, L. Landowski, C. Schiepers, and S. A. Rooholamini, “Picture archiving and communication system-asynchronous transfer

mode network in a midsized hospital," *J Digit Imaging*, vol. 10, no. 3 Suppl 1, pp. 99–102, Aug. 1997, doi: 10.1007/BF03168669.

[9] U. H. Hübner, N. Egbert, and G. Schulte, "Clinical Information Systems – Seen through the Ethics Lens," *Yearb Med Inform*, vol. 29, no. 1, pp. 104–114, Aug. 2020, doi: 10.1055/s-0040-1701996.

[10] W. D. Bidgood, S. C. Horii, F. W. Prior, and D. E. Van Syckle, "Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging," *J Am Med Inform Assoc*, vol. 4, no. 3, pp. 199–212, 1997.

[11] J. Walker, E. Pan, D. Johnston, J. Adler-Milstein, D. W. Bates, and B. Middleton, "The value of health care information exchange and interoperability," *Health Aff (Millwood)*, vol. Suppl Web Exclusives, pp. W5-10-W5-18, 2005, doi: 10.1377/hlthaff.w5.10.

[12] M. W. Woolrich *et al.*, "Bayesian analysis of neuroimaging data in FSL," *Neuroimage*, vol. 45, no. 1 Suppl, pp. S173-186, Mar. 2009, doi: 10.1016/j.neuroimage.2008.10.055.

[13] S. Pieper, B. Lorensen, W. Schroeder, and R. Kikinis, "The NA-MIC Kit: ITK, VTK, pipelines, grids and 3D slicer as an open platform for the medical image computing community," in *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 2006.*, Apr. 2006, pp. 698–701. doi: 10.1109/ISBI.2006.1625012.

[14] W. Schroeder, "The ITK software guide: the insight segmentation and registration toolkit," Jan. 2003, Accessed: Aug. 29, 2023. [Online]. Available: https://www.academia.edu/17857342/The_ITK_software_guide_the_insight_segmentation_and_registration_toolkit

[15] A. Rosset, L. Spadola, and O. Ratib, "OsiriX: an open-source software for navigating in multidimensional DICOM images," *J Digit Imaging*, vol. 17, no. 3, pp. 205–216, Sep. 2004, doi: 10.1007/s10278-004-1014-6.

[16] R. W. Cox *et al.*, “A (sort of) new image data format standard: NiFTI-1,” presented at the 10th Annual Meeting of the Organization for Human Brain Mapping, Jan. 2004.

[17] R. A. Robb and D. P. Hanson, “A software system for interactive and quantitative visualization of multidimensional biomedical images,” *Australas Phys Eng Sci Med*, vol. 14, no. 1, pp. 9–30, Mar. 1991.

[18] M. Jenkinson and S. Smith, “A global optimisation method for robust affine registration of brain images,” *Med Image Anal*, vol. 5, no. 2, pp. 143–156, Jun. 2001, doi: 10.1016/s1361-8415(01)00036-6.

[19] M. Jenkinson, P. Bannister, M. Brady, and S. Smith, “Improved Optimization for the Robust and Accurate Linear Registration and Motion Correction of Brain Images,” *NeuroImage*, vol. 17, no. 2, pp. 825–841, Oct. 2002, doi: 10.1006/nimg.2002.1132.

[20] S. M. Smith, “Fast robust automated brain extraction,” *Hum Brain Mapp*, vol. 17, no. 3, pp. 143–155, Nov. 2002, doi: 10.1002/hbm.10062.

[21] D. S. Rana, G. Hurst, L. Shepstone, J. Pilling, J. Cockburn, and M. Crawford, “Voice recognition for radiology reporting: is it good enough?,” *Clin Radiol*, vol. 60, no. 11, pp. 1205–1212, Nov. 2005, doi: 10.1016/j.crad.2005.07.002.

[22] L. Faggioni, F. Coppola, R. Ferrari, E. Neri, and D. Regge, “Usage of structured reporting in radiological practice: results from an Italian online survey,” *Eur Radiol*, vol. 27, no. 5, pp. 1934–1943, May 2017, doi: 10.1007/s00330-016-4553-6.

[23] P. A. Marcovici and G. A. Taylor, “Journal Club: Structured radiology reports are more complete and more effective than unstructured reports,” *AJR Am J Roentgenol*, vol. 203, no. 6, pp. 1265–1271, Dec. 2014, doi: 10.2214/AJR.14.12636.

[24] V. Granata *et al.*, “Structured reporting of computed tomography in the staging of colon cancer: a Delphi consensus proposal,” *Radiol Med*, vol. 127, no. 1, pp. 21–29, Jan. 2022, doi: 10.1007/s11547-021-01418-9.

[25] T. A. Morgan, M. E. Helibrun, and J. Charles E. Kahn, “Reporting Initiative of the Radiological Society of North America: Progress and New Directions,” *Radiology*, Nov. 2014, doi: 10.1148/radiol.14141227.

[26] G. Shen, X. Jin, C. Sun, and Q. Li, “Artificial Intelligence Radiotherapy Planning: Automatic Segmentation of Human Organs in CT Images Based on a Modified Convolutional Neural Network,” *Front Public Health*, vol. 10, p. 813135, Apr. 2022, doi: 10.3389/fpubh.2022.813135.

[27] J. Becker *et al.*, “Artificial Intelligence-Based Detection of Pneumonia in Chest Radiographs,” *Diagnostics (Basel)*, vol. 12, no. 6, p. 1465, Jun. 2022, doi: 10.3390/diagnostics12061465.

[28] C. A. Mallio, A. C. Sertorio, C. Bernetti, and B. Beomonte Zobel, “Large language models for structured reporting in radiology: performance of GPT-4, ChatGPT-3.5, Perplexity and Bing,” *Radiol Med*, vol. 128, no. 7, pp. 808–812, Jul. 2023, doi: 10.1007/s11547-023-01651-4.

[29] L. C. Adams *et al.*, “Leveraging GPT-4 for Post Hoc Transformation of Free-text Radiology Reports into Structured Reporting: A Multilingual Feasibility Study,” *Radiology*, vol. 307, no. 4, p. e230725, May 2023, doi: 10.1148/radiol.230725.

[30] A. Harmouche, F. Kövér, S. Szukits, T. Dóczi, P. Bogner, and A. Tóth, “WebMRI: Brain extraction and linear registration in the web browser,” *Imaging*, vol. 15, no. 1, pp. 31–36, Jun. 2023, doi: 10.1556/1647.2023.00111.

[31] A. Harmouche, F. Kövér, S. Szukits, T. Dóczi, P. Bogner, and A. Tóth, “XReport: An online structured reporting platform for radiologists,” *SoftwareX*, vol. 17, p. 100993, Jan. 2022, doi: 10.1016/j.softx.2022.100993.

[32] C. Simonyi, M. Christerson, and S. Clifford, “Intentional software,” presented at the Sigplan Notices - SIGPLAN, Oct. 2006. doi: 10.1145/1167515.1167511.

[33] N. Jain, P. Mangal, and D. Mehta, “AngularJS: A modern MVC framework in JavaScript,” *J. Global Research in Computer Science*, vol. 5, pp. 17–23, Jan. 2015.

10. Publications of the author

Total number of articles published by the author: 3
Cumulative impact factor: 3,268
Number of publications that form the basis of this dissertation: 2 **(IF: 3,268)**

10.1. Topic related journal articles

A. Harmouche, F. Kövér, S. Szukits, T. Dóczi, P. Bogner, and A. Tóth, "XReport: An online structured reporting platform for radiologists," *SoftwareX*, vol. 17, p. 100993, Jan. 2022, doi: 10.1016/j.softx.2022.100993.
Quartile: Q2, Impact factor: **2.868** (2023)

A. Harmouche, F. Kövér, S. Szukits, T. Dóczi, P. Bogner, and A. Tóth, "WebMRI: Brain extraction and linear registration in the web browser," *Imaging*, vol. 15, no. 1, pp. 31–36, Jun. 2023, doi: 10.1556/1647.2023.00111.
Quartile: Q4, Impact factor: **0.4** (2023)

10.2. Other articles

G. Jandó, E. Mikó-Baráth, A. Czigler, **A. Harmouche**, I. Szabó, L. Závor, and D. P. Piñero, "Amblyopia screening with the dynamic random dot stereotest," *Ophthalmology Times Europe*, vol. 16, no. 7, Sept. 2022

SoftwareX 17 (2022) 100993

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

XReport: An online structured reporting platform for radiologists

Ahmed Harmouche [a,*], Ferenc Kövér [b], Sándor Szukits [a], Tamás Dóczi [c], Péter Bogner [a], Arnold Tóth [a]

[a] Department of Medical Imaging, Medical School, University of Pécs, 7624 Pécs, Ifjúság str., 13., Hungary
[b] Pécs Diagnostic Center, 7623 Pécs, Rét str, 2., Hungary
[c] Department of Neurosurgery, Medical School, University of Pécs, 7623 Pécs, Rét str, 2., Hungary

ARTICLE INFO

Article history:
Received 26 September 2021
Received in revised form 17 December 2021
Accepted 11 January 2022

Keywords:
Structured reporting
Radiology
eHealth
JavaScript

ABSTRACT

Currently the most widespread way of reporting in radiology is dictation mainly due to performance benefits. The output of this method is plain text, which varies in style (structure, nomenclature, abbreviations, etc.) and content between doctors even when reporting the exact same case. Templated radiology provides a structure for reporting and aims to help in generating more unified reports. We propose a web-based system for creating and using radiological structured reporting templates.

We developed our software based on web technologies. We wrote the system with modular design in mind. We have separate libraries for the different functionalities: a rendering library which renders the templates based on a schema, an editor library which handles template creation, and an evaluator library, which parses, and executes our custom domain specific language, FormScript, which enables dynamic behavior in our templates. We also developed a Single Page Application to create, browse, use and share templating reports. The backend of the application is powered by Firebase from Google.

We deployed our system at a publicly accessible domain at https://app.radiosheets.com.

Code metadata

Current code version	*v1.4.1*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00182
Code Ocean compute capsule	*Not available*
Legal Code License	*MIT License*
Code versioning system used	*git*
Software code languages, tools, and services used	*JavaScript, TypeScript, HTML5, CSS, Bootstrap, Firebase, npm*
Compilation requirements, operating environments & dependencies	*Webpack, Node.js, npm, Angular, Firebase*
If available Link to developer documentation/manual	https://wpmed92.github.io/xreport/
Support email for questions	ahmedharmouche92@gmail.com

Software metadata

Current software version	*1.4.1*
Permanent link to executables of this version	https://app.radiosheets.com
Legal Software License	*MIT License*
Computing platforms/Operating Systems	*Cross-platform, Web-based system*
Installation requirements & dependencies	*No installation needed, works in any modern browser.*
If available, link to user manual - if formally published include a reference to the publication in the reference list	
Support email for questions	ahmedharmouche92@gmail.com

* Corresponding author.
E-mail addresses: ahmedharmouche92@gmail.com (Ahmed Harmouche), ferenc.kover@gmail.com (Ferenc Kövér), szukits.sandor@pte.hu (Sándor Szukits), doczi.tamas@pte.hu (Tamás Dóczi), bogner.peter@pte.hu (Péter Bogner), prsarn@gmail.com (Arnold Tóth).

https://doi.org/10.1016/j.softx.2022.100993

1. Motivation and significance

There is no consensus among radiologists on what a good radiological report is. Both radiologists and clinicians who receive the reports have different views on the optimal layout and content [1]. Currently the most preferred way of reporting is dictation. With the advance of speech recognition technology, it became faster, more accurate and easier to produce reports by dictating than by typing [2]. One of the key features of dictation is that it eliminates context switching. Radiologists do not have to take their eyes off the image at any point during reporting, whereas in case of typing they switch between looking at the keyboard and looking at the screen. However, the problem with both dictation and typing is that the output is plain text. Saving large amounts of reports in the Health Information System (HIS) in plain text will generate an archive which is not maintainable, and not searchable. Valuable information will be lost. Also, as mentioned earlier, no two radiologists will write the same report about the exact same image. The use of different nomenclatures, ordering of findings and abbreviations may result in confusion among doctors, and inefficiency in communication and patient management. To address these issues another form of reporting aroused: structured reporting (SR). SR gives doctors a framework for writing reports. Mostly this framework is template based. A good example of this is the RadReport reporting template collection created by the Radiological Society of North America (RSNA) [3]. The collection contains templates grouped by specialties. The templates are submitted by the users and then are reviewed by the Template Library Advisory Panel to ensure the submissions meet certain criteria. The templates are composed of sections, subsections, and input fields. The report is generated by filling the form. Using such templates for reporting have multiple benefits. They can help give radiologists a guideline on what the report should include for a given pathology or modality thus diminishing the possibility of missing some important findings or information. Furthermore, extracting data from such templates and saving it in a database is straightforward, as opposed to plain text reports.

The drawback to templated radiology is that it is hard to find the optimum of how much a report should be structured. If one tried to cover all the possible cases and logical branches using built-in input fields with predefined options, it would be too time consuming to write the report. But if one used mostly text areas, the structured report would resemble a dictation template, and the benefits of SR would be less significant.

In an online survey members of the Italian Society of Medical Radiology (SIRM) were asked about SR. According to the respondents the most appreciated advantages over conventional reporting were higher reproducibility, better interaction with referring clinicians and the option to link metadata. On the other hand, the most significant disadvantages were risk of excessive simplification, template rigidity, and poor user compliance. Overall, most radiologists asked in the survey were in favor of radiological SR [4]. In a study involving twelve radiology trainees it was shown that SR templates for chest radiographs were statistically significantly more complete and more effective than unstructured reports [5]. Template quality is crucial in maximizing the advantages SR can give. What to include, and what not to include in SR templates should be carefully discussed, and thoroughly reviewed. To indicate the importance of this quality assurance, it is worth noting that there are specialized groups for this task, like the aforementioned Template Library Advisory Panel of RSNA. In a study which aimed to build a CT-based SR template for colon cancer staging, the experts (members of SIRM) used a modified Delphi process to assess a level of agreement for all report sections in the SR template [6].

We propose a new radiological structured reporting software that is free, cross-platform, can be integrated into the dictation-based reporting workflow of a radiologist, enables template creation and report generation. With our solution we aim to make structured reporting more widespread and accessible, thus increasing the quality and consistency of radiological reports.

2. Software description

The software was written as a web application to support all operating systems and devices. Two programming languages were used throughout the development process, namely JavaScript and TypeScript. The project can be divided into two main parts: the library and the application. The library is a standalone module that implements the core features of the software: template building and reporting. The application can be any host that integrates the library, in our case it is a Single Page Application (SPA). Our workflow of creating reporting templates resembles intentional programming [7]. The programmer builds the foundation (template builder) of the software on top of which the domain expert (radiologist) can build the actual application (template). The programmer can later add if-else logic to the template.

The library

The library exposes four public methods to interact with: makeWidget, togglePreviewMode, getReportAsText and getTemplateForUpload. The entry point is makeWidget. It can instantiate a new empty template builder or load a template from a Uniform Resource Locator (URL). Internally it creates an instance of each of the following classes: XReportDOM, XReportRender, Evaluator. The XReportDOM implements a custom subset of the Document Object Model (DOM) which allows only specific elements of the DOM or compositions of DOM elements to be used. The XReportRender calls the render methods of the XReportDOM entities and uses them to assemble either a builder or a viewer component, depending on whether the library is in editor or viewer mode. In editor mode the templates can be modified, whereas in viewer mode they are read-only, and are ready to generate reports. The Evaluator is an interpreter for our Domain Specific Language (DSL) called FormScript. It adds dynamic behavior to the templates through simple if-else logics and calculations. An example of a typical use case for FormScript is to show or hide a specific field if certain conditions are met, or to calculate a score for a scoring system. To view the generated report the library exposes the togglePreviewMode method. Calling this method will transfer the viewer from reporting state to output state or vice versa. In output state the reporter can see the textual output of the form. The generated text can be accessed by the getReportAsText function. When the library is in editor mode a template can be saved by first getting it in JavaScript Object Notation (JSON) format with getTemplateForUpload and then sending it to a web service or storing it locally.

Template structure

The templates are composed of rows, which may have one or more groups in it. Groups are label-entity pairs, and entities are the form's input elements.

The JSON structure of a template is as following:

{ "formScript": "Form script source code is here", report: [{ XFormElem #1 }, { XFormElem #2 }…]}

General fields in XFormElem:

- type: defines what element to render, e.g. row, group, sel, mulsel
- id: a random generated unique identifier

Table 1
Syntactical comparison of FormScript and JavaScript.

Features	JavaScript	FormScript
Logical and	&&	and
Logical or	||	or
Power	**	^
if expression	if (a == b) { … }	if a == b { … }

- scriptAlias: an identifier/variable name by which FormScript can reference the field; auto-generated, but can be changed by user
- hideFromOutput: determines whether the value of the field should be visible in the generated text output
- hidden: determines whether the field should be rendered
- children: a list of groups in a row
- child: the entity of a group

There are fields specific to each entity but they are not listed here.

FormScript

FormScript is a DSL that is specifically designed to run inside XReport templates. It allows custom logic to be executed safely in forms, thus adding dynamic behavior to them. The script can be edited when the library is in editor mode and is accessible through the getScript library call. Once saved, it is stored in the same JSON file as the template itself.

The FormScript syntax is similar to that of JavaScript with some syntactical differences shown in Table 1.

Supported binary operations: addition (+), subtraction (-), division (/), multiplication (*), modulo (%), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal to (==), logical and (and), logical or (or), to the power of (^)

Unary operations: unary not (!), unary minus (-), unary plus (+)

Statements: expression, assignment, if, function call

Types: string, boolean, number

Numerical and string literals are supported. The only variables that are allowed in FormScript are references to form elements. As mentioned earlier, variables are defined in the editor through the scriptAlias property. Function calls are defined only on variables. It is not allowed to declare functions neither are there predefined library functions without an element context. Calling a function has the following form: variable.function(…parameters). Functions should be defined for XFormElem classes. When XReport loads a report, it checks for an attached script. If there is a script attachment, it will start running it in an Evaluator instance.

Backend

Our SPA has a backend powered by Google Firebase to store the template resources. We store the template files in storage buckets. The metadata for each template, such as date of creation, creator's username, template name, template category is saved to Cloud Firestore documents.

The process of uploading a template to our backend includes the following steps:

- query template JSON from the library through getTemplateForUpload
- assemble upload metadata: date of creation, category, username, template name, template URL
- save the metadata to a Cloud Firestore document
- upload the template JSON file to the storage

Frontend

The frontend is built as a SPA using the Angular [8] and the Bootstrap Cascading Style Sheets (CSS) frameworks. Every icon used in the app are taken from the Font Awesome icon library. Angular supports client-side navigation, asynchronous data binding among others, which enables us to easily fetch and render views. To retrieve templates from Cloud Firestore we use the official Firebase JavaScript Software Development Kit (SDK) and the RxJS reactive programming library. In Firebase terms the templates form a collection, and individual entries in this collection are documents. To show these documents on the screen we followed the Model-View-ViewModel pattern with data binding.

3. Illustrative examples

Template viewing and building

The viewer/builder page is where we load our templates and render them with our library as show in Fig. 1. The form is centered horizontally and have a slight drop shadow around it. There is a button group on the right side of the form which contains different buttons based on which state the page is currently in (viewer or builder).

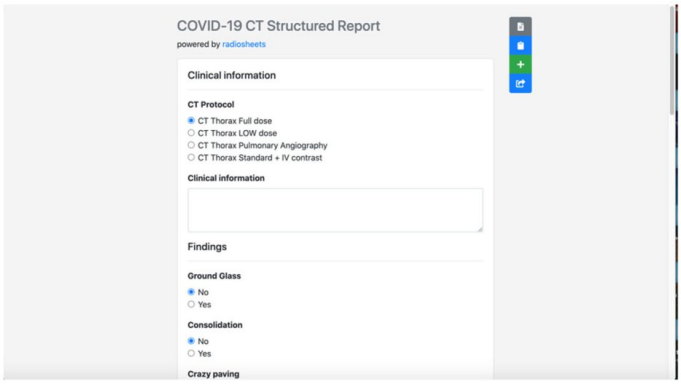

Fig. 1. A COVID-19 CT template rendered in XReport.

Buttons in viewer state:

- preview report
- copy to clipboard (copies the report output to clipboard)
- new report
- share (copy the template link to clipboard so that it can be shared)

Buttons in builder state:

- save template
- discard template

Builder components

There are 13 components to choose from when building a template:

- Text field
- Plain text
- Number field
- Calculated field
- Boolean field
- Single choice
- Multiple choice
- Textarea
- Date
- Header
- Information
- Rating scale
- Image

If we take an oncological example, a question regarding the size of the tumor would be a number field, a TNM staging system could be created using single choice fields, or a rating scale, etc. Every component is added to the form with a label attached. A component-label combination is called a group. Groups are added to rows, and rows are added to sections. The sections make up the whole template.

Component editors

Every component has an editor view as shown in Fig. 2. Every component type has its own editable properties. For example, the input field has a unit property (mm, cm, etc.), a single choice field has an options property, an image has an URL property. The component editor is activated by hovering the mouse over the component, then clicking on the pencil icon. Components can be deleted by clicking on the minus sign.

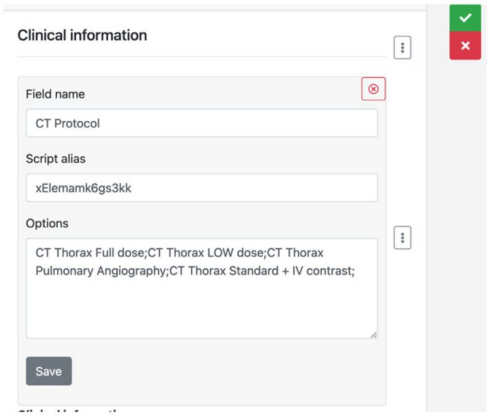

Fig. 2. A component editor for a single choice input field.

Row editors

Row operations can be performed by clicking on the three vertical dots at the end of each row. The click event will trigger a secondary menu to open with all the components, and two actions: delete and duplicate.

FormScript editor

On the main builder component there is a button with a branch icon which toggles the view between template editing and FormScript editing. The FormScript editor as shown in Fig. 3 is a simple resizable text area where the user can edit the dynamic logic that is attached to the template. When switching back from script editing, the script is automatically evaluated and the changes are visible.

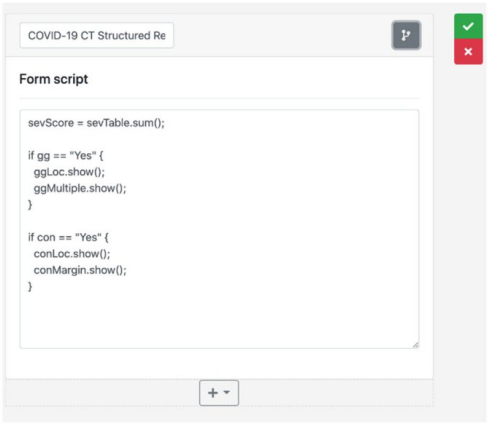

Fig. 3. The FormScript editor.

4. Impact

With XReport we built a free, cross-platform structured reporting platform for radiologists. It enables both creating and viewing reporting templates in an easy, user-friendly way. We built our software with modular design in mind and refactored the core features into a separate library to make embedding it into other products easy. We also built an application with the library embedded in it to demonstrate the easy integration. Furthermore, we designed a simple DSL called FormScript to add dynamic logic to our forms. The main feature of it, and the reason we created it in the first place, is security. It does not allow malicious code executions unlike the eval function of JavaScript. It is also very simple to use because of its limited feature set. Our templates are dynamic, responsive and have modern design. The templates generate easy to copy–paste structured textual output to be compatible with any HIS, and to integrate well into dictation-based workflows. Our templates help not only in precise reporting, but also serve as a guide for radiologists thanks to our custom form elements such as images and rating tables.

We compared our solution to a similar free service developed by RSNA. From a technological point of view both programs are similar since they are built using web technologies but they have their differences when it comes to the ecosystem, editing process and user experience. The RSNA template library has a

more mature ecosystem: there are a lot of contributors who build and upload templates, there are some nice to have features such as favouriting a template. But the template editing itself is less advanced than ours. In the RSNA editor the screen flow to get to the actual editing is as following: click on "Create and Upload a Template button", click on "T-Rex Template Editor", interact with a popup which asks how the user wants to start the editing, click on one of the options. In our program the screen flow is a lot simpler: click on "Add new template", and you are in the editor. In the RSNA editor adding individual elements has some issues. The elements have to be drag and dropped from a side panel, which is problematic on mobile devices as there is not enough space. The element editor works as a pop-up which brings the user out of the editing context. In our app adding elements is responsive (works on mobile devices as well), and is inline, so the user remains in the editing context throughout the whole process. When it comes to how dynamic the templates are we found that RSNA templates do not allow dynamic behavior such as hiding/showing elements based on certain conditions. Through FormScript our system enables fully dynamic behavior. The RSNA editor lacks some important elements such as images and rating tables which are essential in information sharing and oncological grading systems.

5. Conclusions

This paper introduces XReport, a free, web-based structured reporting platform for radiologists. It enables both creating and viewing reporting templates in an easy, user-friendly way. Our system is deployed at https://app.radiosheets.com and is ready to be used. Template creation and editing requires login, but template viewing, copying the generated reports and sharing the templates do not. The templates currently available in the app have been created and are used by our research group and by radiologists from Pécsi Diagnostic Center and from the University of Pécs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

A.T. was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

References

[1] Ganeshan D, et al. Structured reporting in radiology. Academic Radiol 2018;25(1):66–73.
[2] Rana DS, et al. Voice recognition for radiology reporting: is it good enough? Clin Radiol 2005;60(11):1205–12.
[3] Morgan TA, Helibrun ME, Kahn CE. Reporting initiative of the radiological society of north america: Progress and new directions. Radiology 2014;273(3):642–5.
[4] Faggioni L, Coppola F, Ferrari R, Neri E, Regge D. Usage of structured reporting in radiological practice: results from an Italian online survey. Eur J Radiol 2017;1934–43.
[5] Marcovici PA, Taylor GA. Journal Club: Structured radiology reports are more complete and more effective than unstructured reports. AJR Am J Roentgenol 2014;203(6):1265–71.
[6] Granata V, Faggioni L, Grassi R, et al. Structured reporting of computed tomography in the staging of colon cancer: a Delphi consensus proposal. La Radiol Med 2021;126(2):181–8.
[7] Simonyi C, Christerson M, Clifford S. Intentional software, In: ACM Sigplan Notices; 2006.
[8] Jain N, Bhansali A, Mehta D. Angularjs: A modern MVC framework in JavaScript. J Glob Res Comput Sci 2014;5(12):17–23.

AKADÉMIAI KIADÓ

IMAGING

ORIGINAL ARTICLE

WebMRI: Brain extraction and linear registration in the web browser

AHMED HARMOUCHE[1*], FERENC KÖVÉR[2], SÁNDOR SZUKITS[1], TAMÁS DÓCZI[3], PÉTER BOGNER[1] and ARNOLD TÓTH[1]

[1] Department of Medical Imaging, Medical School, University of Pécs, Pécs, Hungary

[2] Pécs Diagnostic Center, Pécs, Hungary

[3] Department of Neurosurgery, Medical School, University of Pécs, Pécs, Hungary

Received: December 22, 2022 • Revised manuscript received: March 6, 2023 • Accepted: March 17, 2023

ABSTRACT

Background and Aim: Since the initial release of the World Wide Web, the capabilities of web browsers have grown from presenting formatted documents to running complex programs, such as 3D game engines. The medical imaging community started to adopt technologies that came with the fifth major version of the HyperText Markup Language (HTML5). It led to the creation of various web-based radiological applications such as cornerstone.js or BrainBrowser. BrainBrowser supports both 3D and 2D rendering of neuroimaging data. However, it cannot run important image processing algorithms, such as brain extraction and linear registration, which are essential in most neuroimaging workflows. The most commonly used library that supports these algorithms is the FMRIB Software Library (FSL). We aim to build a web-based cross-platform neuroimaging platform that combines data visualization with image processing.

Methods: We built our system as an extension of BrainBrowser. We developed WebMRI in JavaScript and designed the user interface using HTML, CSS, and Bootstrap. We used Emscripten to port the brain extraction and linear registration tools of FSL to the web.

Results: We built WebMRI, a fully web-based extensible neuroimaging platform that combines the visualization capabilities of BrainBrowser with the brain extraction and linear registration tools of FSL by porting them from C++ to WebAssembly. We extended BrainBrowser with a plugin system that makes it easy to bring other processing algorithms into the platform. We released the WebMRI source code on Github: https://github.com/wpmed92/WebMRI.

Conclusions: We developed and released WebMRI, a web-based cross-platform open-source neuroimaging platform.

KEYWORDS

neuroimaging, MRI, image processing, JavaScript, WebAssembly

Introduction

The web was created to facilitate information-sharing between universities and institutes. Initially, it was a medium to exchange simple documents with links and basic formatting. After decades of development and innovation, modern browsers can play high-quality video streams, complex animations, games, and simulations and perform other computationally intensive tasks natively without using any third-party plugins. The possibility of plugin-free native application execution inside browsers came with the 2014 release of the asm.js specification [1]. Asm.js was introduced as a strict subset of JavaScript to be used as a low-level target language for compilers. It allows the browsers to perform ahead-of-time optimizations on the code and to run it with superior performance to unoptimized JavaScript. The de-facto compiler toolchain for asm.js is Emscripten [2]. It can compile C/C++ code

IMAGING 15 (2023) 1, 31–36
DOI: 10.1556/1647.2023.00111

*Corresponding author.
Tel.: +36 30/883 84 35.
E-mail: ahmed.harmouche@aok.pte.hu

bases by converting Low Level Virtual Machine (LLVM) [3] bitcode to asm.js. An early demonstration of the viability and ease of use of the toolchain was the porting of Unreal Engine 3 to the web, which was developed as a collaboration of Mozilla and Epic Games in 4 days [4]. The successor to asm.js is WebAssembly, which first appeared in 2017 [5]. It has similar goals to asm.js, most importantly to allow running computationally intensive programs in the browser, but instead of being a subset of JavaScript, it is a binary instruction format for a stack-based virtual machine. WebAssembly is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications. Emscripten supports WebAssembly too, and changing the compilation target to WebAssembly instead of asm.js is straightforward.

Besides games and multimedia software, medical applications also benefited from modern-day web innovations. Although medical software systems, such as Health Information Systems (HIS), have been on the web for a long time, it was only after the introduction of the fifth major version of the HyperText Markup Language (HTML5) [6] the Web Graphics Library (WebGL) [7] and WebAssembly that computationally more involved medical programs, such as radiological software tools, started to appear on the web.

One such example is cornerstone.js [8], a framework that allows viewing Digital Imaging and Communications in Medicine (DICOM) [9] images inside the browser. It consists of a collection of libraries, each responsible for a specific task, like decoding DICOM images, connecting to a Picture Archiving and Communication System (PACS), or annotating medical images. For medical education-related use-case, Biodigital Human [10] is worth mentioning. It is an interactive 3D platform where one can explore, customize, and share 3D views of human anatomy, physiology, diseases, and treatments.

The neuroimaging community also started to adopt web technologies, an example of which is BrainBrowser [11], a web-based project that enables visualizing surface and volumetric data. The WebGL-based Surface Viewer is capable of rendering 3D surfaces in real-time, whereas the Volume Viewer uses HTML5 canvas to allow slice-by-slice traversal of volumetric data in Neuroimaging Informatics Technology Initiative (NIfTI) and various other neuroimaging formats.

Post-processing of neuroimaging data is often needed for accurate visualization. As an example, in the case of the simultaneous visualization of two MRI volumes of the same patient acquired at different times, overlaying the two on top of each other might not be sufficient because of the slightly different positions of the patient during the two image acquisitions. Linear registration of the two volumes is needed so that the same location on the two images refers to the same voxel (Fig. 1). This way, the two images can be overlaid on top of each other for more accurate evaluation. For linear registration to work robustly, another processing step has to be performed on the two input volumes. This is called brain extraction, which is the method of deleting non-brain tissue from an image of the whole head.

The most widespread tools for performing the aforementioned image processing techniques on neuroimaging data are the Brain Extraction Tool (BET) [12] and the FMRIB Linear Image Registration Tool (FLIRT) [13, 14] of the FMRIB Software Library (FSL) [15]. Another widely used software to perform linear registration is Slicer 3D [16], which focuses more on a graphical user interface-based approach, whereas the components of the FSL library are mainly used as command-line tools. Slicer 3D is also capable of performing brain extraction, however it is available as an extension and is not part of the software by default.

To the best of our knowledge, there is no fully client-side web-based neuroimaging workflow that supports brain extraction and linear registration. Our goal is to create a web-based, cross-platform, extensible neuroimaging platform that works in any modern web browser and does not require a web server for the computations. We aim to combine the visualization capabilities of BrainBrowser VolumeViewer, and the image processing power of FSL by porting FSL BET and FLIRT to WebAssembly and integrating them into BrainBrowser through an extensible plugin system.

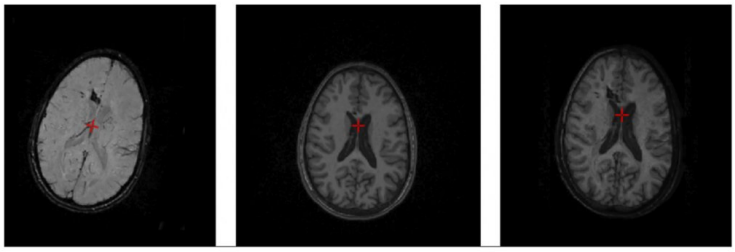

Fig. 1. Linear registration – An SWI MRI volume (first image) is linearly registered to a reference T1-weighted MRI volume (second image). Since the patient position was different during the image acquisitions, the registration algorithm has to calculate a transformation matrix that aligns the input volume to the reference volume. The final image shows the corrected input volume superimposed on the reference volume

Methods

Software architecture

Our project consists of three main parts:

- the porting of the brain extraction (BET) and linear registration (FLIRT) FSL tools to WebAssembly
- the integration of the ported tools into BrainBrowser through a plugin system
- creating a demo application to showcase multiple neuroimaging workflows using the ported tools

We created a version of FSL that can run on the web using a tool called Emscripten. Our approach involved customizing the software by selectively including only the components required for executing BET and FLIRT. These changes allowed us to generate WebAssembly programs that can be used to run BET and FLIRT in a web browser. To enhance user experience, we used web workers to run the tools, this way the browser window remains responsive while the program performs the computation. From a user perspective, the ported tools behave exactly like their native command-line versions, i.e., they can receive and parse the same command-line arguments, and they generate output files in the same manner.

We integrated our ported tools into BrainBrowser (Fig. 2), which allows visualization of NIfTI volumes through its VolumeViewer module.

Software functionalities

By default, BrainBrowser supports loading files in NIfTI, Medical Imaging NetCDF (MINC), and Massachusetts General Hospital (MGH) formats, but we extended it with

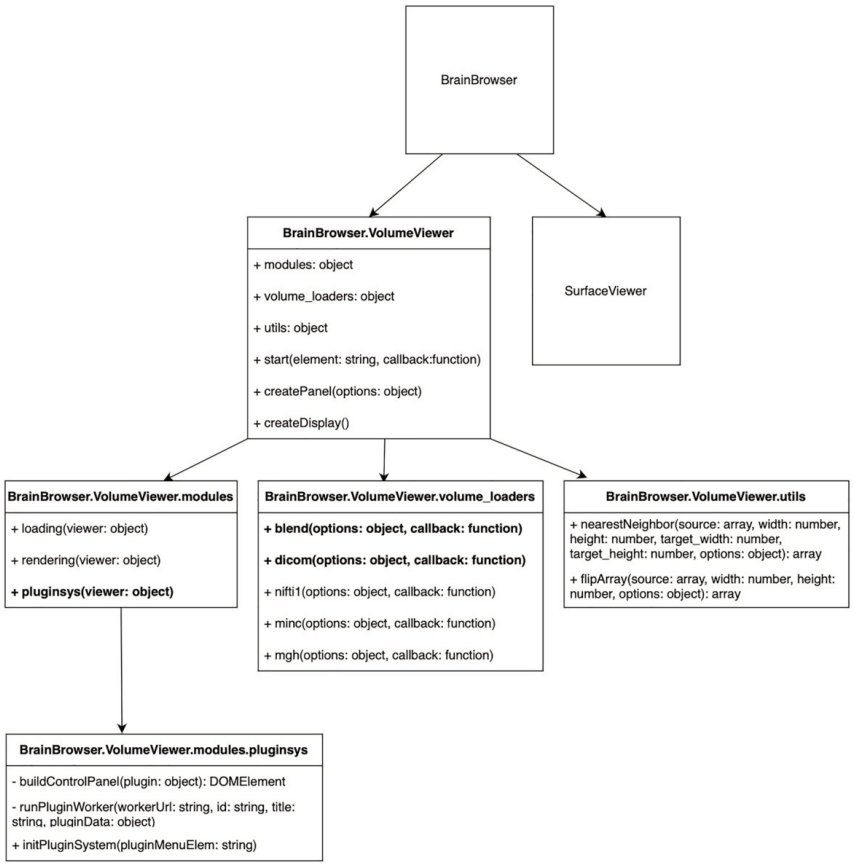

Fig. 2. Extending BrainBrowser - WebMRI defines a plugin system on top of BrainBrowser VolumeViewer, and adds two additional volume loaders: "blend" and "dicom". The plugin system handles automatic GUI generation ("buildControlPanel") and runs the web workers ("runPluginWorker") that invoke the WebAssembly binaries

two additional volume types: "dicom" and "blend". The "dicom" loader allows DICOM slices to be loaded into the program by internally performing a DICOM to NIfTI conversion and then calling the NIfTI loader using the output of the conversion step as its input file. For the conversion, we used an Emscripten port of an open-source tool named dcm2niiX (the port is based on version v1.0.20170207). The "blend" loader allows overlaying two volumes on top of each other, which helps visualize two volumes that are linearly registered to each other.

To allow running our ported tools in BrainBrowser, we extended it with a plugin system. The plugin system is defined as a VolumeViewer module.

Plugins are defined as an array of plugin objects. Each object describes a specific plugin with "name", "title", "author", "id", "worker," and "gui" attributes. The "author" property stores a link to the website of the author of the plugin, in our case, FSL. The "id" is a unique identifier for the plugin. The "worker" is a link to the web worker that invokes the WebAssembly module, and the "gui" is a link to the JavaScript Object Notation (JSON) file that describes the user interface of the plugin, which shows tweakable parameters and a button to run the tool. When the user clicks on the "Run" button on the plugin dialog, the UI handler loops through the input fields and creates name-value pairs, which in turn are passed to the web worker. The web worker will then run the WebAssembly program with the received command-line arguments.

Results

The entry point of our demo application showcasing WebMRI is the dashboard (Fig. 3). There is a navigation bar on the top with menu items "File", "Tools", and "About". The "File" menu hosts actions such as opening DICOM series, a NIfTI volume, or creating a blended volume ("Create overlay") out of two input volumes. The "Create overlay" functionality is an important last step in a linear registration workflow, as it simultaneously visualizes the two volumes registered to each other. The "Tools" menu shows the list of the plugins available in the application (BET and FLIRT), and through the "About" menu, users can access the developer documentation and user manual of WebMRI.

Under the navigation bar, we present the main screen, which is divided into the rendering section on the right, and the panel section on the left. The rendering section hosts the BrainBrowser VolumeViewer widget, which visualizes the loaded volume in sagittal, axial, and coronal planes. The panel section next to it hosts the "Workspace" and "Volume controls" widgets.

The "Workspace" widget (Fig. 4) shows the files currently opened in the application. These files can either be loaded by the user (input files) or generated by plugins (output files). The workspace has a root folder, which is a collection of files that are visible to plugins as input files and plugin-specific folders, into which plugins generate their output files. If the users want to further process output files, they first have to move the relevant files to the root folder by clicking on the blue up arrow next to the file names. This way, the selected files will be visible to the plugins. The "Volume controls" widget contains coordinate information, blending controls, and windowing sliders.

Our demo application showcases our two ported FSL tools. They can be accessed from the "Tools" menu. Figure 5 shows the automatically generated user interface for BET. The input volumes can be selected from a drop-down menu. The user is presented with a list of modifiable parameters,

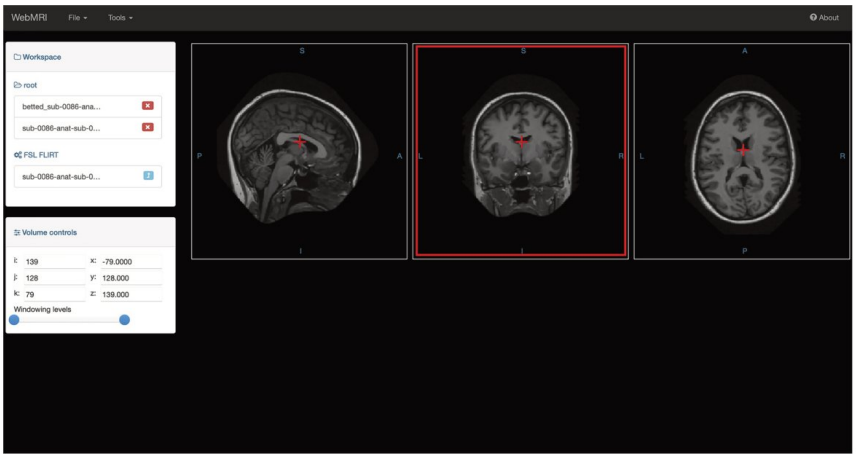

Fig. 3. The WebMRI dashboard - The image shows the demo application screen with the navigation bar on top, the panel section on the left, and the rendering view on the right. The rendering view is provided by BrainBrowser VolumeViewer. The red crosshair shows the currently selected voxel, and the coordinates are shown in the "Volume controls" widget

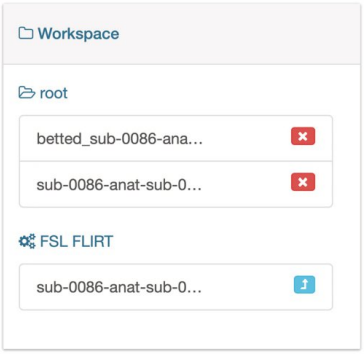

Fig. 4. The workspace widget - It contains the main "root" folder and the plugin-specific folders. Volumes can be moved to the "root" folder and can be deleted. Only volumes in the "root" folder are visible to the plugins

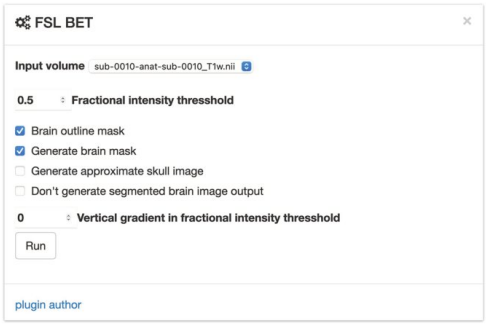

Fig. 5. Plugins – Automatically generated plugin UI for the ported FSL BET brain extraction tool

which will translate to command-line arguments by the plugin system when the "Run" button is clicked, and the underlying tool is invoked.

The most practical use case our system is capable of supporting in its current form is the multi-modal assessment of neuroimaging data. As an example, let us look at how cerebral microbleeds can be detected in an MRI volume. The most widespread way of analyzing microbleeds is to leverage susceptibility weighted imaging (SWI). Although SWI can help identify microbleeds with high sensitivity, their localization is challenging relying solely on this modality. T1-weighted MRI images preserve the anatomical features of the brain, so using the T1 modality in combination with SWI can help with localization. WebMRI can co-register an SWI to a T1 MRI volume thus enabling precise microbleed detection. To enhance registration result brain extraction can also be performed. Since our system can handle DICOM to NIfTI conversion, the only step the user has to perform before loading data into our system is to export the DICOM series from the PACS.

Discussion

With WebMRI, we built a cross-platform, extensible neuroimaging platform that supports brain extraction and linear registration and can be run in any modern web browser without using third-party browser plugins or a dedicated web server. For brain extraction, we used the BET, and for linear registration, the FLIRT tool of the popular neuroimaging library collection FSL. For the porting process, we used Emscripten to convert the native programs to WebAssembly binaries. We extended BrainBrowser with a plugin system to be able to load our ported tools into the system combining volumetric visualization with image processing. We built a demo application that incorporates the ported tools and demonstrates that a complete neuroimaging workflow from DICOM loading to linear registration can be performed inside a web browser.

Building on the foundations of BrainBrowser, we increased the scope of browser-based neuroimaging and showed that not only neuroimaging data visualization but computationally intensive processing algorithms could also be performed inside modern web browsers. In contrast to Slicer 3D (to which our system is similar in terms of modularity) or other not browser-based programs, our solution does not require an installation. By removing the need for a web server, and complex setup processes, we reduced the cost and improved the ease of use of certain neuroimaging workflows. These factors can improve the clinical adoption of new image-processing tools. Although we demonstrated the porting of two popular neuroimaging tools, there is no limit to bringing other algorithms into the WebMRI plugin system as long as they are written in a programming language that can be compiled to WebAssembly. As more and more programming languages used in computationally intensive application development (such as Rust) support compilation to WebAssembly, the number of libraries that can be ported to WebMRI will increase.

The limitation of our fully browser-based system is twofold. The memory usage of heavy workloads, such as processing large numbers of volumes in bulk, might exceed the resources available inside a browser tab. There is also a performance penalty for the porting due to the overhead of the WebAssembly runtime compared to fully native execution. The impact of these limitations varies between programs and browsers and can be mitigated using optimizations in the Emscripten toolchain or the original codebase. If these factors make it not possible to reliably run a given tool in the browser, it can still be run in a web server since Emscripten supports execution in a Node.js runtime.

In the future, we plan to increase our fleet of ported neuroimaging tools and make our plugin system more modular and extensible. We aim to make it easier to

integrate our system into a clinical PACS environment, so there will be no need to export DICOM files. We also plan to support hybrid execution so that a given algorithm can be run on the client or server side, depending on the requirements of the given workload. We plan to improve the visualization capabilities of WebMRI by adding support for customizable volume viewer widgets and annotation of points of interest on the volumes.

Ethical statement: The study submitted to IMAGING have been conducted in accordance with the Declaration of Helsinki and according to requirements of all applicable local and international standards.

Authors' contribution: AH – Conceptualization, Software architecture design, Software development, Software testing, Drafting of the manuscript, Preparing the Figures. FK – Conceptualization, Software testing, Revision. SSz – Conceptualization, Software testing, Revision. TD – Conceptualization, Software testing, Revision. PB – Conceptualization, Software testing, Revision. AT – Conceptualization, Product design, Software testing, Revision. All authors reviewed the final version of the manuscript and agreed to submit it to IMAGING for publication.

Funding sources: No financial support was received for this study.

Conflict of interests: The authors have no conflict of interest to disclose.

ACKNOWLEDGMENTS

Not applicable.

REFERENCES

[1] Herman D, et al.: "asm.js" asm.js, 18 August 2014, http://asmjs.org/spec/latest/. Accessed 12 October 2022.

[2] Zakai A: Emscripten: An LLVM-to-JavaScript compiler. OOPSLA '11: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 22 10 2011: 301–12.

[3] Lattner C, Adve V. LLVM: a compilation framework for lifelong program analysis & transformation. In: Proceedings of the international symposium on code generation and optimization, CGO '04, Palo Alto, California, Mar 2004.

[4] Mozilla, and Epic Games: 'Epic citadel' demo shows the power of the web as a platform for gaming - future releases.The Mozilla Blog, 2 May 2013, https://blog.mozilla.org/futurereleases/2013/05/02/epic-citadel-demo-shows-the-power-of-the-web-as-a-platform-for-gaming/. Accessed 12 October 2022.

[5] Haas A. et al.: Bringing the web up to speed with WebAssembly, Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017.

[6] Web Hypertext Application Technology Working Group: HTML Standard. HTML, 2004, https://html.spec.whatwg.org. Accessed 12 October 2022.

[7] Leung C, Salga A: Enabling WebGL. Conference: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26–30, 2010.

[8] Hafey C: Cornerstone.js. Cornerstone.js, 2017, https://www.cornerstonejs.org. Accessed 12 October 2022.

[9] Bidgood WD, Jr., Horii SC, Prior FW, Van Syckle DE: Understanding and using DICOM, the data interchange standard for biomedical imaging. J Am Med Inform Assoc 1997; 4(3): 199–212.

[10] Qualter J, Sculli F, Oliker A, Napier Z, Lee S, Garcia J, et al.: TheBioDigital human: a web-based 3D platform for medical visualization and education. Studies in Health Technology and Informatics 2012; 173: 359–61.

[11] Sherif T, Kassis N, Rousseau MT, Adalat R, Evans AC: Brainbrowser: Distributed, web-based neurological data visualization. Front Neuroinf 2015; 8: 89.

[12] Smith SM: Fast, robust automated brain extraction. Human Brain Mapping November 2002; 17(3): 143–55.

[13] Jenkinson M, Smith SM: A global optimisation method for robust affine registration of brain images. Medical Image Analysis 2001; 5(2): 143–56.

[14] Jenkinson M, Bannister PR, Brady JM, Smith SM: Improved optimization for the robust and accurate linear registration and motion correction of brain images. NeuroImage 2002; 17(2): 825–41.

[15] Woolrich MW, Jbabdi S, Patenaude B, Chappell M, Makni S, Behrens T, et al.: Bayesian analysis of neuroimaging data in FSL. NeuroImage 2009; 45: S173–86.

[16] Pieper S, Lorensen B, Schroeder W, Kikinis R: The NA-MIC Kit: ITK, VTK, Pipelines, Grids and 3D slicer as an open platform for the medical image computing community. Proceedings of the 3rd IEEE International Symposium on Biomedical Imaging: From Nano to Macro 2006; 1: 698–701.