UNIVERSITY OF PASSO FUNDO

INSTITUTE OF EXACT SCIENCES AND GEOSCIENCES

GRADUATE PROGRAM IN APPLIED COMPUTING

Decision Support System for Agrotechnology Transfer: a cross-platform tool

Jonas De Abreu Resenes

Passo Fundo

2019

UNIVERSITY OF PASSO FUNDO

INSTITUTE OF EXACT SCIENCES AND GEOSCIENCES

GRADUATE PROGRAM IN APPLIED COMPUTING

DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER: A CROSS-PLATFORM TOOL

Jonas De Abreu Resenes

Thesis submitted to the University of Passo Fundo in partial fulfillment of the requirements for the degree of Master in Applied Computing.

Advisor: Prof. Dr. Willingthon Pavan

Co-Advisor: Prof. Dr. Carlos Amaral Holbig

Passo Fundo

2019

CIP – Catalogação na Publicação

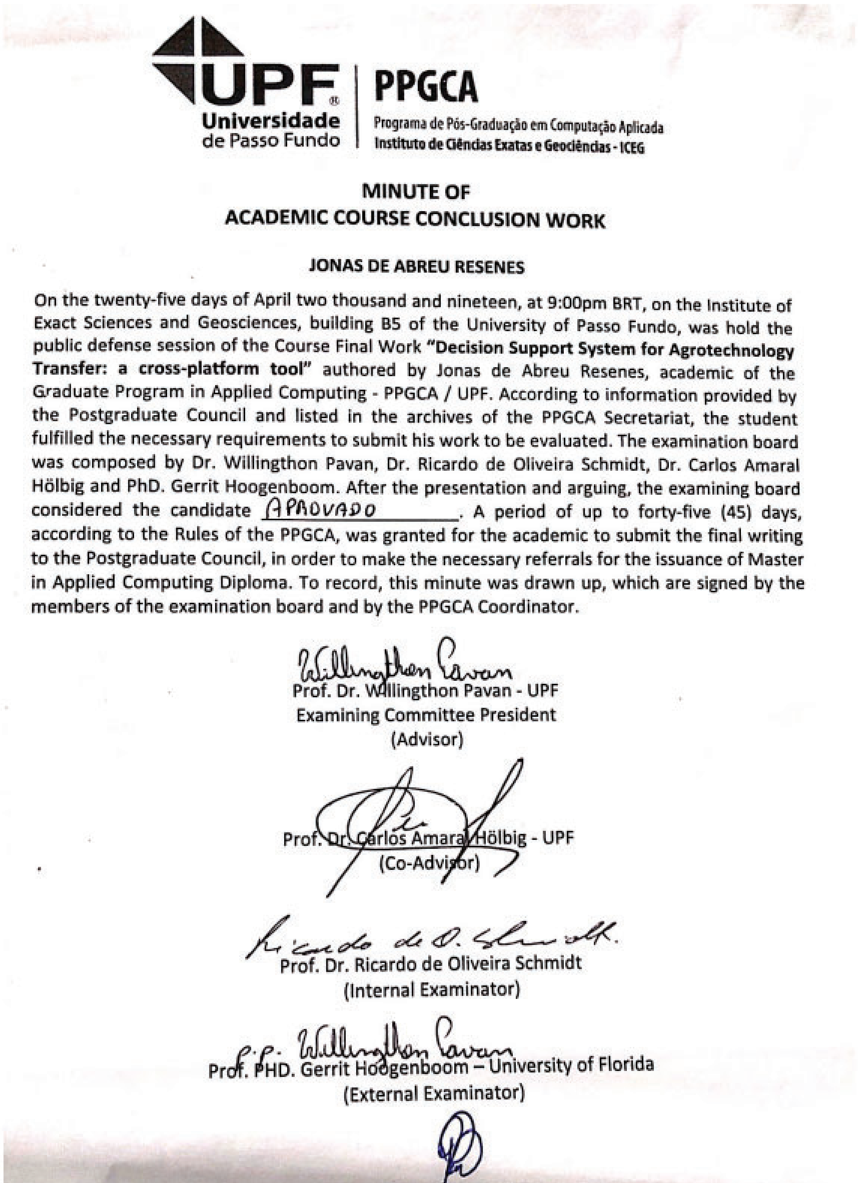
R433d Resenes, Jonas De Abreu
Decision support system for agrotechnology transfer: a cross-olatform tool / Jonas De Abreu Resenes. – 2019.
15 f. : il. ; 30 cm.

Advisor: Prof. Dr. Willingthon Pavan.
Co-Advisor: Prof. Dr. Carlos Amaral Holbig.
Thesis (Master in Applied Computing) – University of Passo Fundo, 2019.

1. Software engineering. 2. Simulation models. 3. Decision making. 4. Web technologies. 5. Agrotechnology. I.Pavan, Willingthon, advisor. II. Holbig, Carlos Amaral, co-advidor. III. Título.

CDU: 631:004

Catalogação: Bibliotecária Marciéli de Oliveira – CRB 10/2113



UPF® Universidade de Passo Fundo | PPGCA
Programa de Pós-Graduação em Computação Aplicada
Instituto de Ciências Exatas e Geociências - ICEG

MINUTE OF
ACADEMIC COURSE CONCLUSION WORK

JONAS DE ABREU RESENES

On the twenty-five days of April two thousand and nineteen, at 9:00pm BRT, on the Institute of Exact Sciences and Geosciences, building B5 of the University of Passo Fundo, was hold the public defense session of the Course Final Work **"Decision Support System for Agrotechnology Transfer: a cross-platform tool"** authored by Jonas de Abreu Resenes, academic of the Graduate Program in Applied Computing - PPGCA / UPF. According to information provided by the Postgraduate Council and listed in the archives of the PPGCA Secretariat, the student fulfilled the necessary requirements to submit his work to be evaluated. The examination board was composed by Dr. Willingthon Pavan, Dr. Ricardo de Oliveira Schmidt, Dr. Carlos Amaral Hölbig and PhD. Gerrit Hoogenboom. After the presentation and arguing, the examining board considered the candidate APROVADO. A period of up to forty-five (45) days, according to the Rules of the PPGCA, was granted for the academic to submit the final writing to the Postgraduate Council, in order to make the necessary referrals for the issuance of Master in Applied Computing Diploma. To record, this minute was drawn up, which are signed by the members of the examination board and by the PPGCA Coordinator.

Willingthon Pavan
Prof. Dr. Willingthon Pavan - UPF
Examining Committee President
(Advisor)

[illegible]
Prof. Dr. Carlos Amaral Hölbig - UPF
(Co-Advisor)

[illegible]
Prof. Dr. Ricardo de Oliveira Schmidt
(Internal Examinator)

p.p. Willingthon Pavan
Prof. PHD. Gerrit Hoogenboom – University of Florida
(External Examinator)

[illegible]
Prof. Dr. Rafael Rieder
PPGCA Coordinator

ACKNOWLEDGMENTS

First of all, I would like to thank my wife, Luana Schneider, for all support during this journey.

Many thanks for my Adivisor Prof. Dr. Willingthon Pavan and my Co-Advisor, Prof. Dr. Carlos Amaral Holbig for all the support and feedbacks. Also, thanks Prof. Dr. José Mauricio Cunha Fernandes for the feedbacks and partnership during this work.

Special thanks for Dr. Gerrit Hoogenboom for invite me to join on the 10th DSSAT sprint.

DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER: UMA FERRAMENTA MULTIPLATAFORMA

RESUMO

O DSSAT-CSM, simula o crescimento e desenvolvimento de uma cultura ao longo do tempo. O uso simulação de modelos de culturas pode auxiliar na tomada de decisões, permitindo a definição das melhores práticas de gestão do campo. A plataforma DSSAT contém um conjunto de aplicativos que foram desenvolvidos usando linguagens de programação que estão no fim do ciclo de suporte e disponibilizado no ambiente Windows somente. O conjunto de ferramentas do DSSAT realiza operações comuns entre elas, como leitura de arquivos e integração com o DSSAT-CSM, porém, cada aplicação tem sua forma de integração, o que torna a manutenção e evolução dessas ferramentas uma tarefa árdua. Desta forma, o objetivo deste trabalho é criar um padrão de integração com o DSSAT-CSM que facilite a criação de um aplicativo multiplataforma para Windows, Linux e MacOS para executar modelos de simulação do DSSAT-CSM. Como resultado deste trabalho obteve-se um módulo JavaScript projetado para facilitar e integração com o DSSAT-CSM em diferente plataformas fornecendo um estilo moderno de desenvolvimento de software; DSSAT-Lite uma ferramenta para executar simulações DSSAT sobre o protocolo HTTP, que fornece a capacidade de executar e visualizar simulações por meio da web; DSSAT-Shell, uma ferramenta multiplataforma que possibilite a execução de modelos de cultura do DSSAT. Deste modo, o conjunto de tecnologias desenvolvidas torna-se relevante para outros desenvolvedores e membros da comunidade científica do DSSAT que queiram criar ferramentas de integração com DSSAT usando as tecnologias de ponta na área de desenvolvimento de software.

Palavras-Chave: modelos de simulação, tomada de decisão, tecnologias web.

DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER: A CROSS-PLATFORM TOOL

ABSTRACT

The DSSAT-CSM simulates the growth and development of a crop over time. The use of crop simulation models can help in decision making, allowing the definition of the best field management practices. The DSSAT platform contains a set of applications that are in the end of life support and are available in the Windows environment only. The DSSAT applications performs common operations, such as reading files and integrating with DSSAT-CSM, however each application has its own way of integration, which makes the maintenance and evolution of these tools an arduous task. Thus, the objectives of this work is to create a integration standard with DSSAT-CSM that can facilitate the creation of a multiplatform application for Windows, Linux and MacOS with the objective of executing simulation models. As a result of this work, we obtained a JavaScript module designed to facilitate the integration with DSSAT-CSM in different platforms providing a modern software development style; DSSAT-Lite a tool to perform DSSAT simulations over the HTTP protocol, which provides the ability to perform and view simulations through the web; DSSAT-Shell, a cross-platform tool that enables the DSSAT simulation models. This way, the set of standars and applications developed becomes relevant to other developers and members of the DSSAT community to help them on the DSSAT-CSM integration as well as developing new DSSAT tools.

Keywords: simulation models, decision making, web technologies.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. INTRODUCTION

Crop simulation models are capable of predicting final yield and have been intensively studied in various parts of the world [1]. These simulation models are computer programs that represent mathematically the growth of plants in relation to the environment [2]. To predict possible agricultural scenarios, several crop models have been developed over the years. These models have several objectives, among them, to predict harvest yield of a crop, to represent the growth simulation and to predict risks of diseases in a tillage [3].

The creation of multidisciplinary groups worldwide, many simulation models of crop growth and development have been developed for the most varied purposes [4]. Among the models developed stands out the DSSAT (Decision Support System for Agrotechnology Transfer). It is a system composed of several simulation models, process-oriented, designed for global applications and working independently of location, station and cultivation [5].

DSSAT is a set of computer programs for simulating agricultural crop growth, it combines crop simulation models (CSMs), database management programs and decision support systems (DSSs) [6]. The DSSAT includes a range of tools to assist users in preparing datasets for running large or complex simulations, and for analysing results [7]. These tools were built in programming languages such Visual Basic and Delphi also these tools are only supported in Windows OS.

The DSSAT-CSM was developed to facilitate the application of crop systematics in agronomic research. Its initial development was motivated by the need to integrate soil, climate, crop and management knowledge into making the best decisions on the transfer of production technology from one location to another where soil and climate are different [8]. Different types of applications are accomplished in DSSAT-CSM by using different modes of operation [9]. The modes of operation are accessed through commands.

With every technological advance and the diffusion of the use of computer programs, new computational challenges arise. Replacing the command line interface with graphical interface has facilitated the increasing use of programs by users. However, this transformation has caused another problem for software developers such reuse code in different platforms and integration with different programming language. Also, reuse of software components is becoming more import in variety of aspects of software engineering [10], specially because the fact that many system contains similar or even identical compontents that are developed from scratch over and over again.

In order to reduce the effort of developing new tools and also expand the existing ones, this work aimed to create a standard and reusable interface between DSSAT-CSM and other tools in different development platforms. The interface will be used to create a expanded DSSAT Shell to run on Windows, Linux and MacOS as well as the Web.

Considering the diversity of the objectives presented and also describe the development, this work is divided into the following phases. Chapter 2, a background of literature review, concepts and technologies used in this work as well as a review about DSSAT structure. Chapter 3, addresses how the jDSSAT module was create and how it can be used for DSSAT-CSM integration. Chapter 4 presents the development of a lite version of the DSSAT-Shell to run over the web. Chapter 5 addresses the cross-platform approach applyed to build a new desktop version of the DSSAT-Shell. Finally, the chapter 6, final considerations and conclusions of this work.

2. LITERATURE REVIEW

2.1 MODELING AND SIMULATION

Simulation, according Shannon [11], is a process of creating a model to test behaviors and evaluate strategies variables, in a criteria limit, for real systems. The term **Simulation** can be divided into two main categories: a) computational, which are those that need a computer to be performed; b) non-computational, those that do not require a computer to be performed, for example, a designer using a reduced prototype of an aircraft in a wind tunnel to simulate behavior of a real aircraft [12].

Notably, when we think about simulation, we tied it to simulate the behaviour of some system. Forrester [13] in his work, says that a system is a grouping of parts that operate together, aiming at a common goal. According to Forrester definition [13], a system assumes a cause-and-effect interaction between the parts that compose it, so that iterations between these parts are clearly identified with the purpose of the system.

Usually, when we imagine a system, we associate our thoughts with a real system, in other words, that exists physically. There is also the possibility of hypothetical systems, which does not exist physically. To create a validation of these systems it is necessary to construct simulation models, which are an abstraction of reality, which approaches the truly behavior of the system, but always simpler than the real system [12]. There are three basic categories of simulation models:

- **SYMBOLIC MODELS**: a symbolic model, is composed of graphic symbols that represent a system in a static way. A process flowchart can be considered as a symbolic model [12]. The symbolic models are used mainly in project documentations and as a communication tools.
- **MATHEMATICAL MODELS**: these models can be identified as a set of mathematical formulas, such as linear programming models. For the most part, these models are static in nature and do not provide analytical solutions for complex systems [12].
- **SIMULATION MODELS**: simulation models are used in real systems. A simulation model can capture more accurately characteristics of random variables, trying to repeat in a computer the same behavior that the system would present when submitted to the same boundary conditions [12]. The simulation model is used particularly as a tool to obtain answers to sentences such as: "What happens if ...".

Many agricultural crop simulation models are applied to strategic and tactical management decision-making in the same way as income forecasting. The expected variation in crop yield and related variables, as well as the use of natural resources, is mainly due to the short and long term variation of climatic conditions. The results produced by the models can be used to make appropriate management decisions and to provide farmers alternative options for their farming system [14].

Due to the formation of researchers groups, many simulation models and development of cultures were developed for the most varied purposes. This work highlights one of these models, the DSSAT (Decision Support System for Agrotechnology Transfer).

2.1.1 Decison Support System for Agrotechnology Transfer (DSSAT)

Decison Support System for Agrotechnology Transfer (DSSAT) is a collection of database, models and different applications that are operated by software that helps the user to select, compare different options and analyze the final results for different crop models [15]. Its v4.7 version is made up of more than 42 types of crop simulation models [16].

DSSAT has been used for more than 30 years by researchers around the globe [16]. It was developed by the IBSNAT (International Benchmark Sites Network for Agrotechnology Transfer) project at the University of Hawaii from 1974 [8], whose goal was to accelerate the flow of agrotechnology and increase the success rate of technology transfer agricultural research centers for agricultural fields [5]. The IBSNAT project lasted ten years, where its activities resulted in the creation of DSSAT [8] and in a global network of collaborators in diverse activities such as the development, testing and application of models to solve problems in the agriculture around the world [17].

The DSSAT development was done using different programming languages and with little attention to the structure of *software*. It has been redesigned to facilitate the incorporation of new scientific advances, applications, documentation and maintenance. The principle of this remodeling is a modular structure, called **CSM** (cropping system model), in which the components are separated along scientific discipline lines and structured to allow easy replacement or addition of new modules in the DSSAT-CSM [8]. The Figure 1, shows that DSSAT-CSM is made up four different components: a) a database collection, used to organize and store a minimum set of input data required to run crop models; (b) a series of crop models; c) applications to analyze and display the simulated experiments; d) an user interface [5].

Each DSSAT module has six operational steps, as shown in Figure 2, which are responsible for structuring and organizing a simulation, such as: initialization, crop installation, rate calculations, integration, daily outputs and summary of outputs. The *Main Program* controls when each of these steps is triggered and when each module should perform a task.

In 1998, Jones [17] explains the different operations that can be carried out in DSSAT, which are: a) entry, organization and storage of data on culture, soil and climate; b) query, ana-

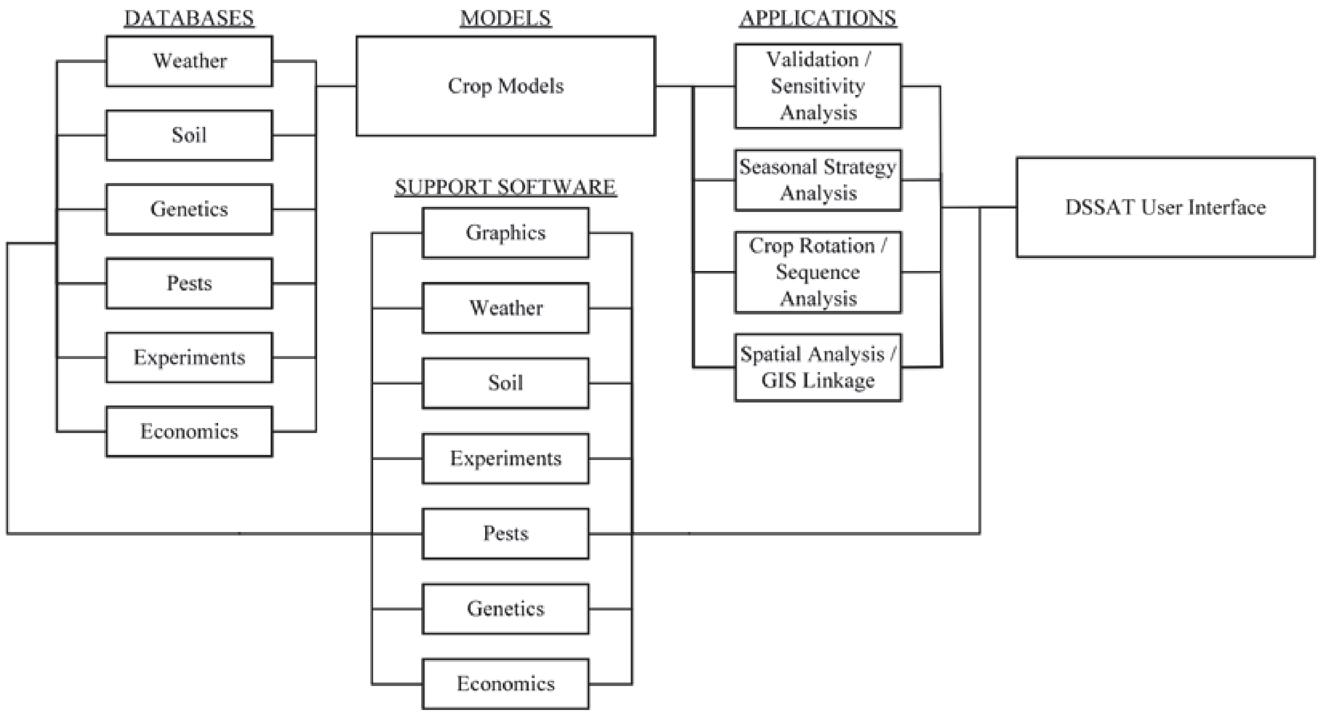


Figure 1. Diagram of database, application, and support software components and their use with crop models for applications in DSSAT [8].

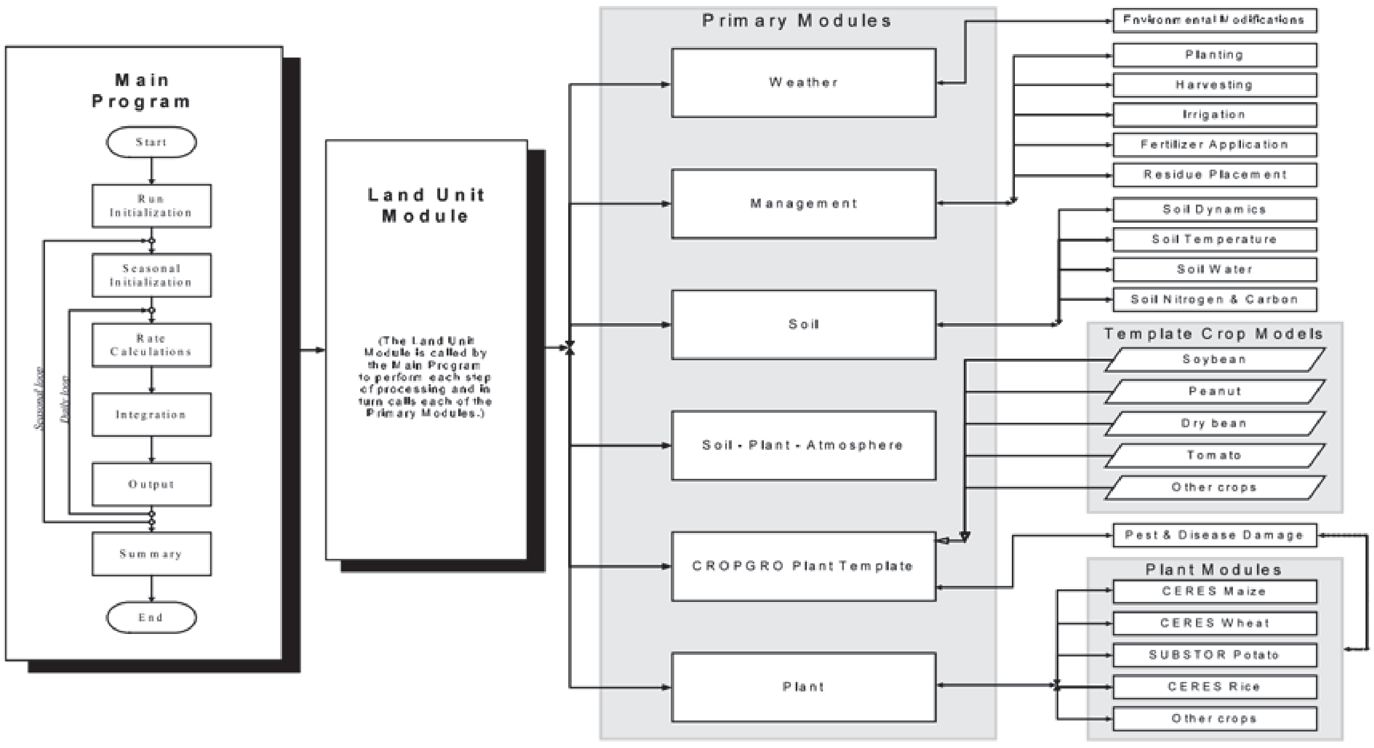


Figure 2. Overview of the components and modular structure of the DSSAT-CSM [8].

lyze and display data; c) calibrate and evaluate the growth of crop models; d) evaluate different management practices of a field.

The programs that execute the DSSAT functions were written in various programming languages [5]:

- Fortran (modelos);
- C (shell);
- Object Pascal (shell);
- Visual Dbase (shell) and
- Basic (strategy and risk management programs).

DSSAT consists of several programs that assist users in creating new experiments[18], são eles:

- XCREATE, the purpose of this feature is to enable the creation of new experiment description files.
- WEATHERMAN, the proposal of this program is to enable a reformatting of climate data files to make them compatible with DSSAT v3 models, to insert new data, to generate climatic data, to calculate statistics and to provide graphs about climatic data.
- WINGRAF, is used for graphic simulation and data observation, which can be obtained after the simulation of a crop model;
- GENCAL, the *Genotypic Coefficient Calculator*, enables the user to calculate crop coefficients for different crops. This program is responsible for rotating the crop models for an existing experiment in a given number of times.
- LIST/EDIT, this DSSAT functionality enables users to obtain multiple lists of different data types. For example, this function can list all available experiments in a directory.
- SOIL PROFILE CREATE, the proposal of this program is to enable the user to create new soil profiles or new profiles for their experiments.

The DSSAT functions have been developed primarily to support the use of crop simulation models in decision making [17]. The following crop models are currently accessible in the DSSAT environment:

- CERES for cereal models, which include corn, wheat, rice, barley and sorghum..
- CROPGRO de models for legumes such as soybeans and peanuts.
- CROPSIM for root crops such as cassava and potato.

- CROPGRO allows users to modify values in a model culture file without changing any code.

One of DSSAT's strengths is the ability to analyze many management strategies. Several years of simulation are run using climate data generated to simulate the effects of the climate in the future [8]. In addition, DSSAT is suitable for long-term studies, which assists in the evaluation of efficient crop management strategies and optimize production [19].

Among the limitations of the DSSAT, the restriction to homogeneous field-scale analyzes stands out [8]. Also, the fact that crop models are written using the FORTRAN programming language makes it difficult to integrate with new components. The DSSAT user interface, Figure 2, is supported on Windows only and uses technologies in the end of life support.

2.2 MULTIPLATFORM DEVELOPMENT

Currently, there are many different operating systems available to computer users. The development for each of these operating systems is a rather arduous task. All operating systems provide API [1] to facilitate the development and access to the peripherals in their platform, however, there is no standard among the operating systems. So, the portability problem of computer programs comes into play. The purpose of this section is to review the main frameworks and tools available to assist in the development of multi-application applications.

2.2.1 Multiplatform framework

A multiplatform framework is made to accelerate the development of applications that must be available in different platforms. In this sense, multiplatform frameworks aid developer to write a single code base that is capable of being compiled for different platforms. Many of these frameworks use web technologies so the code can be reused.

2.2.2 Standard Web Development Technologies

2.2.2.1 HTML5

HTML5 is the latest evolution of the standard that defines HTML (HyperText Markup Language), which is the most basic block builder of the Web. It is used to describe and define the content of a web page. HTML is made up of elements where each of these can be modified by defined attributes of the specification itself. HTML elements have attributes, these will configure or adjust the operation and behavior of the element in many ways. The basic structure of an HTML document can be seen in Figure 3.

[1] set of routines and programming standards

```
<!DOCTYPE html>
<html>
<head>
<title>Title</title>
<!-JavaScript and CSS Dependencies->
<meta charset='utf-8'>
</head>
<body>
<!-Markup declarations->
</body>
</html>
```

Figure 3. A Simple HTML Documen. Hypertext Markup Language (HTML) is the most common language used to create documents on the World Wide Web. HTML uses different tags to define a layout for web pages. Most tags require an opening <tag> and a closing </tag>.

Some HTML5 specifications have become quite useful for the development of cross-platform applications, they are:

- **Connectivity**, allows communication with the server in modern and innovative ways.
- **Offline and storage**, allows web pages to store data locally on the client side and operate offline more efficiently;
- **2D/3D Graphics and effects**, enables a wide range of graphic representation options.
- **Performance and integration**, provides great speed optimization and better use of computer hardware.
- **Device access**, enables the use of various input and output methods and devices.

2.2.2.2 CSS

CSS (Cascading Style Sheets) is a style language used to describe the presentation of a document written in HTML or XML (eXtensible Markup Language). CSS is one of the main languages of Open Web [2] and has been standardized by W3C [20]. CSS files are created with the extension`*.css` and are declared inside an HTML document, as can be seen in Figure **??**.

2.2.2.3 JavaScript

JavaScript is a lightweight language, interpreted, object-based language with first-class functions, that is, treats classes as if they were functions of the language itself. It was created

[2]movement of cooperation and standardization of web development

in 1995 by Brendan Eich, an American computer programmer former chief of technology office at Mozilla Corporation. JavaScript is also known as the scripting language for web pages. In addition, it is a multi-paradigm and dynamic language, supporting object-oriented, imperative and functional styles [21].

ECMAScript is the specification of the scripting language that JavaScript implements, it is the formal and structured description of a script-based language, being standardized by Ecma International, an association created in 1961 dedicated to the standardization of information and communication systems, in the ECMA-262 specification. The most recent version of the specification is ES6, created in 2015, also called ECMAScript 2015.

In addition, there are popular JavaScript *engines*. These are responsible for interpreting the script generated on the browser side. Are they:

- **V8**, used by the Google Chrome browser and in the latest versions of the Opera browser [22].
- **JavaScriptCore** used in some browsers WebKit [23] as Apple Safari.
- **Carakan** used in older versions of the Opera browser [24].
- **Chakra** used by Internet Explorer [25].

The JavaScripts files have extension `*.js` and are declared inside an HTML document, as can be seen in Figure 3. An alternative to declaring files directly in HTML documents is the use of WebPacks, section 2.2.2.6, which is widely used in modern Web development.

2.2.2.4 React

React is a library for creating interfaces, which was idealized by Facebook. In essence, it is based on components, meaning, this library aims for maximum reuse of code. One of the advantages of using React is to use Virtual DOM [3], which makes accessing the components of a page very fast when compared to access DOM [4] page.

2.2.2.5 Angular

Angular is an open source JavaScript library that is sponsored and maintained by Google. It has been used in some of the largest and most complex web apps around [26]. Angular applications are built around a design pattern called Model-View-Controller (MVC), which places an emphasis on creating applications that are:

[3]DOM representation in the JavaScript language
[4]tree with the HTML components of a native

- Extendable: It is easy to figure out how even a complex Angular app works once you understand the basics—and that means you can easily enhance applications to create new and useful features for your users.
- Maintainable: Angular apps are easy to debug and fix, which means that long-term maintenance is simplified.
- Testable: Angular has good support for unit and end-to-end testing, meaning that you can find and fix defects before your users do.
- Standardized: Angular builds on the innate capabilities of the web browser without getting in your way, allowing you to create standards-compliant web apps that take advantage of the latest features (such as HTML5 APIs) and popular tools and frameworks.

2.2.2.6 Webpack

The webpack is a code packer for web projects. Its main objective is to focus on modules of an application. The webpack facilitates the independent development of software, it allows you to split the code into multiple modules to be read on demand, which makes applications load faster.

When the webpack processes an application, it recursively creates a dependency graph that includes all the modules an application needs, and then packages all of those modules into a small number of packages, often just one, to be loaded by the browser. The starting point of this graph is known as the entry point, it is the contextual root or the first file loaded by the application. The entry point tells the webpack where to start and follows the dependencies graph to know what to group.

2.2.3 Frameworks and tools for cross-platform development

One of the challenges of cross-platform software development is how an interface would interact in different platforms natively. In this sense, a study will be presented on the main frameworks that have a set of API's that help the developer in this task.

2.2.3.1 Node.js

Node.js is a JavaScript code interpreter based on the Google Chrome V8 [22] engine, which works on the server side. It was invented in 2009 by Ryan Dahl, and developed using C and C ++, where one of its main goals was to assist programmers in developing applications with high scalability.

The Node.js architecture works around a main thread where all events that happen on the server side are monitored. It follows non-blocking I/ O standards [5] due to the fact that the framework is asynchronous in nature. One of the main aspects of Node.js is the event handling, it provides a specific module for handling events, the module events, this module provides a generic class for event handling, the EventEmitter class [27].

Also, Node.js provide a set of modules for operating system integration, that aid applications developed with web technologies to access file system and other OS tools.

2.2.3.2 npm

NPM (Node Package Manager) is the Node package manager, and one of the most widely used open source repository. The main features of NPM are:

- Online repository for publishing open source projects for Node.js.
- Command-line utility that interacts with this repository online that helps package installation, version management and dependency management.

2.2.3.3 Travis-ci

Travis is a software distribution framework integrated with Github [6]. Once a build is triggered, travis clones the repository from GitHub into a new virtual environment and performs a series of tasks to create and test the code of the application. If the build does not generate any errors, travis can deploy the code to any application server.

Builds can also automate other parts of the delivery workflow. This means that there are tasks that dependens on each other, with build steps, configuration notifications and preparation of deployments after builds. In addition, Travis' continuous integration (CI) enables a development team to create continuous software deliveries, for instance, each new code published in Github, travis compiles it for all the predefined platforms in its settings file.

2.2.3.4 Electron

Electron was created by the Github team and became known as Atom Shell. Electron was created using technologies like Node.js and Chromium, and it is currently behind several projects, such as the Atom editor, Slack and Visual Studio Code.

[5] refers to code that does not block while executing
[6] https://github.com

The Electron architecture is divided into two parts. The main process, which basically shows the graphical user interface creating web pages. The second part called "The Renderer Process" is the process by which the web page uses the multi-processing architecture of Google Chromium to show its interface. This is the fron-end part.

2.2.3.5 Chromium Embedded Framework

Some of the frameworks that support cross-platform development use the CEF (Chromium Embedded Framework). The CEF is an open source project distributed under the BSD license [28] founded by Marshall Greenblatt in 2008 and based on the Google Chromium project [29]. It focuses on facilitating the browser's encapsulation in third-party applications. CEF isolates the user from the underlying complexity of Chrome and blink code. Most features in CEF have standard implementations that offer rich functionality, while requiring little or no user integration work.

CEF supports a wide range of programming languages and operating systems and can be easily integrated into new and existing applications. It was designed from the ground up with performance and ease of use as primary goals. The underlying framework includes programming interfaces using the C and C++ languages, exposed through native libraries that isolate applications from the Chromium implementation details.

2.3 RELATED TOPICS

2.3.1 pyDSSAT

PyDSSAT is a library developed using the Python [7] programming language, which contains a set of scripts that initialize, execute, and process DSSAT models 2.1.1. It was developed with the main goal of agricultural research based on the DSSAT crop modeld and for research in different fields such as climate, meteorology, ecology and hydrology involving agriculture [30]. It was developed as part of the final project for the course APC524 (Software Engineering for Scientific Computing) at Princeton University in the American state of New Jersey [31].

The pyDSSAT library includes some enhancements to command-line execution, including the ability to control incoming files, parse output files with Matplotlib [8] resources, and provide a GUI (Graphical User Interface) tool for efficient and interactive work. In particular, pyDSSAT wraps the Fortran code in an accessible class [31] using f2py [9].

[7]https://www.python.org/ python
[8]https://matplotlib.org/ matplotlib
[9]https://docs.scipy.org/doc/numpy-dev/f2py/ f2py

In order to install pyDSSAT, a valid DSSAT license is required and we need to install *gfortran* [10]. In addition, we should clone the pyDSSAT repository from Git, on the same path where the DSSAT is installed. The complete list of dependencies can be seen in Table **??**.

Table 1. pyDSSAT dependencies (Adapted from pyDSSAT Documentation [31])

Dependency	Link to download
Git	http://git-scm.com/http://git-scm.com/
Python: 2.5 or higher	https://www.python.org/https://www.python.org/
NumPy: 1.2 or higher	http://www.numpy.org/http://www.numpy.org/
Matplotlib	http://matplotlib.org/index.htmlhttp://matplotlib.org/index.html
IPython:	http://ipython.org/http://ipython.org/

After cloning the pyDSSAT code and changing the settings in DSSAT to compile pyDSSAT, it is possible to run DSSAT simulation models in a terminal, for this, it is necessary to specify the crop type, weather station information, simulation start year, simulation end year, plant month and date.

The pyDSSAT API, Figure 4, contains three main steps in its flow: a) preparation of input archives; b) execution of a DSSAT model through the `DSSATLIBRARY.Model` class from DSSAT; c) the post execution that will return the result files of the experiment.

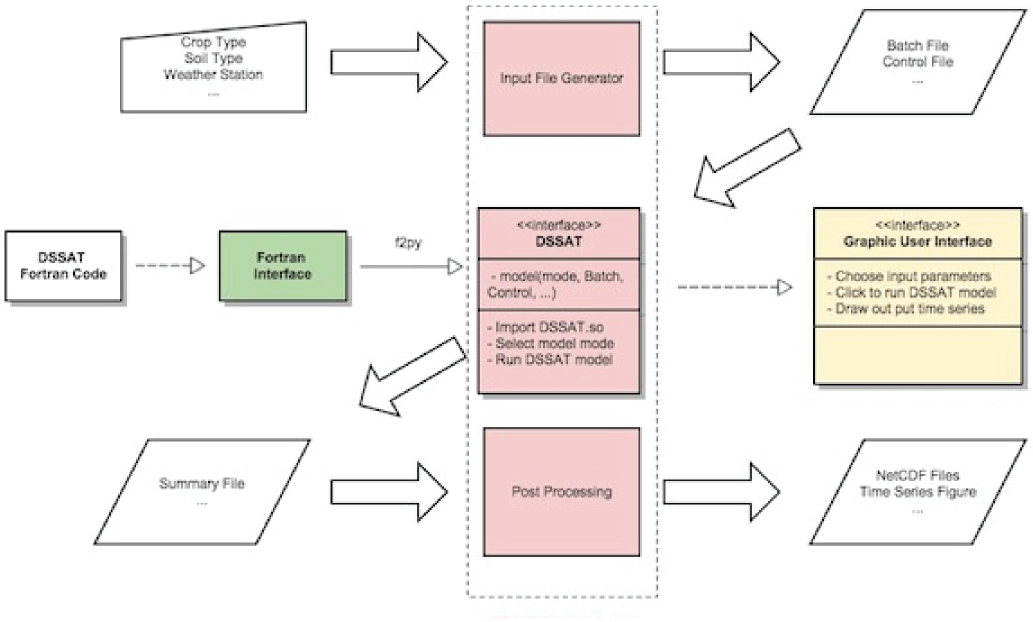


Figure 4. pyDSSAT API diagram [32].

The pyDSSAT's benefits is the use of an interface to run simulations instead of executions using command-line. In other hand, it has a lot of complex configuration that requires some experience from the user. Also, it we were not able to run it on other operating systems such MacOs and Linux.

[10]https://gcc.gnu.org/wiki/GFortran GFortran

3. JDSSAT: A JAVASCRIPT MODULE FOR DSSAT-CSM INTEGRATION

3.1 ABSTRACT

The DSSAT is a collection of computer programs and tools integrated into a single software package in order to facilitate the application of crop simulation models in research and decision making. The DSSAT Shell, an user interface program, enables users to easily select and use any of the DSSAT components. It reads text files, both input and output with fixed width format, to provide information to the users and to be able to run the models. The logic to read DSSAT files and process the information to display to the user relies on the Shell itself and cannot be reusable by any other system, which makes it harder to implement alternatives for the DSSAT Shell since there are no frameworks available that implements the complexity of processing DSSAT files. The DSSAT tools were built using old programming technologies such Visual Basic which is in end-of-life support and Delphi, these technologies should be replaced for a modern and standardized software development approach for a better maintainability. Besides, these tools are stand-alone and they don't share code which increases the effort to maintain them. This work presents the jDSSAT, a multiplataform JavaScript module. The jDSSAT provides a standard and reusable approach for reading and processing DSSAT files. Through this approach, we isolate the complety of processing DSSAT files to allow DSSAT integration on any environment. It also integrates with DSSAT-CSM to make it easier to run DSSAT models in Linux, Windows and MacOS. As a result, we present a multiplatform user interface prototype created to run DSSAT crop models using the main features of the jDSSAT. Also, the integration with the R environment that expands the possibilities of the DSSAT integration.

3.2 INTRODUCTION

Simulation of crop systems has significantly advanced over the past 40 years [33]. Crop modeling can facilitate researchers' ability to understand and interpret experimental results, and to diagnose yield gaps [34]. It can also be useful as a means to help the scientist define research priorities. Using a model to estimate the importance and the effect of certain parameters, a researcher can observe which factors should be more studied in future research, thus increasing the understanding of the system [35].

Decision support tools are designed to help users make more effective decisions by leading them through clear decision stages and presenting the likelihood of various outcomes resulting from different options [36]. These can be dynamic software tools, whose recommendations vary according to the user's inputs, and they may suggest an optimal decision path.

Decision Support Systems for Agrotechnology Transfer (DSSAT) were developed by the International Benchmark Sites Network for Agrotechnology Transfer (IBSNAT) scientists' group [15]. DSSAT are the combination of crop simulation models (CSMs), database management programs and decision support systems (DSSs). CSMs are the computer programs that simulate crop growth and yield based on previously established calculations and input data of soil, weather and crop management practices [6]. The DSSAT-CSM can simulate yield on a range of crops and has been used by many scientists, decision-makers, and researchers all over the world for more than two decades. It is designed to facilitate the application of crop models in a systematic approach to agronomic research composed of more than 42 models. Prior to the development of the DSSAT, crop models were available, but these were used mostly in labs where they were created [8] and there were no sets of programs and a model suite that could be used as a tool for decision-making as well as predicting one or more crops within a cropping system [37].

The DSSAT Shell program provides a user-friendly working environment in which various stand-alone tools and applications are seamlessly integrated with the DSSAT crop models. Within the shell, the user can launch applications for creating and modifying data files, running the crop models, and analyzing the results [38]. However, the DSSAT Shell and other applications of the DSSAT system, which are installed separately, were built using technologies under end-of-life support such Visual Basic. Another point is that the DSSAT tools, including the Shell, are available for Windows OS only, which requires users from Linux and MacOS to deal with the command line to run model in DSSAT-CSM. This approach requires a very good understanding of DSSAT-CSM commands that might be painful for some users.

Other tools like *pyDSSAT* [39], python library to execute original Fortran program on Linux, were built to improve DSSAT's users experience over the terminal command line execution in Linux OS, including the ability of controlling input files, analyzing output files with Matplotlib tools, and providing a GUI toolkit for efficient interactive work integrate with DSSAT-CSM. There is also DASST, which is a R package for reading, processing and writing DSSAT files [40]. It uses tools available in R for statistical and graphical analyses. This package tends to simplify the post processing of DSSAT simulated values stored in .OUT, files offering methods that expose these data as belonging to a collection of data.frame objects that can be thought like tables. However, either pyDSSAT or DASST are not able to provide the same functionalities as DSSAT Shell does. They also do not provide a multiplatform approach and they were not built to provide alternatives to create a modern DSSAT Shell.

This study is motivated by the need to apply software engineering techniques to build a reusable approach for DSSAT integration. We have implemented common functionalities in a JavaScript module to allow any user interface easily read and processes DSSAT files. Tools like DSSAT Shell reads files only what is needed to provide info to the user and to be able to run the model. The implementation of the files processing relies on DSSAT Shell code itself, so it can not be used anywhere else. Any code that does anything with a user interface should only involve user interface code [41]. There is also a need to implement software engineering best practices such cross-platform principle, to allow DSSAT users easily run and visualize simulations from Windows, Linux and MacOs.

The jDSSAT is a JavaScript Module created to be standard approach for DSSAT integration. The module is capable to process different DSSAT files such CDE, OUTPUT, EXPERIMENTS and TREATMENTS. It is designed as a module to either run on a client [11] or backend side [12] and facilitates developers from DSSAT community to create alternatives for DSSAT Shell, without having to worry about interoperability and extra tools installation. The jDSSAT module is cross-platform [13] and provides a series of JavaScript functions for DSSAT-CSM integration that allow any developer to build their own user interface to run DSSAT model in Linux, Windows and MacOS.

The remainder of this paper is structured as follows. Section 2 presents the key technical requirements that guided the jDSSAT development. Section 3 presents the jDSSAT design and architecture. Section 4 covers the main jDSSAT features and their implementation details. Section 5 describes usage patterns for jDSSAT functionalities. Finally, Section 6 presents our conclusions and directions for future research.

3.3 GOALS AND REQUIREMENTS

There are four primary functional requirements that have guided jDSSAT implementation:

1. **Portability:** The jDSSAT should run the same code base in different operational systems, it is the crucial issue for development cost reduction. The operational systems supported are Linux, MacOS, and Windows. Also, it should integrate with DSSAT-CSM to run models on the operating system supported without the developer having to specify it.
2. **Reconfigurability:** To avoid having to rewrite pieces of code when a new DSSAT version is released, we would design a configurable system that can be applied in various scenarios. Below is a summarized list of the settings options for jDSSAT:
   - DSSAT work directory
   - DSSAT versions supported
   - Output files supported
3. **Web-based:** The jDSSAT module may be accessed over HTTP where processing is done over the internet on an external server. A REST interface should be provided to allow the integration with different programming languages.

[11] requests pages from the Server, and displays them to the user. In most cases, the client is a web browser or an offline application

[12] responsible for serving pages

[13] is computer software that is implemented on multiple computing platforms

4. **Reusability:** Reuse of jDSSAT artifacts in various formats. It should be used as "plug-and-play" in the client side using web technologies through NPM and in the server side through REST API.

The jDSSAT is designed to return data in data structures that are well-organized and ready to use, such as objects, lists and vectors. We describe the data structures for jDSSAT as follows:

- Vectors to represent lists with single values such crops available in DSSAT.
- Complex objects containing property name and property value.
- Array of objects as lists of complext objects such output and experiment files.

3.4 THE JDSSAT MODULE DESIGN

The jDSSAT architecture is optimized and straightforward as possible. The module follows facade software-design pattern, Figure 5, to provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use [42]. The facade pattern provides more isolated funcionaties implementation and reduce the risk of adding issues to the sub-systems that are not being changed. This pattern is particularly used when a system is very complex or difficult to understand because the system has a large number of interdependent logic.

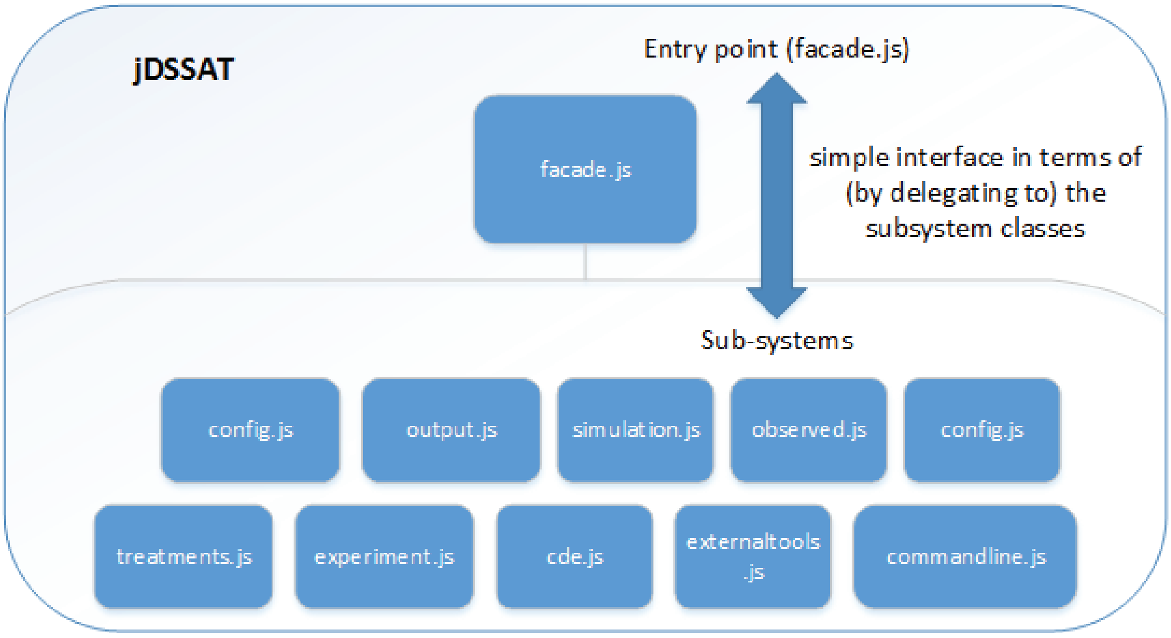


Figure 5. Overview of jDSSAT using Facade design pattern. These entry point access the system on behalf of the facade client and hide the implementation details.

The jDSSAT runs in a Node enviroment. In the Figure 6, we show the architecture components in a user interface context. The user interface loads jDSSAT as dependency and it provides info to the user through jDSSAT. Then, jDSSAT loads NodeJS [43] modules to access the File System, Child Process and Operational System. For instance, if the user interface needs to display the experiments available for a given crop, jDSSAT will process the EXPERIMENT file within crop folder and return a formatted object to the user interface.

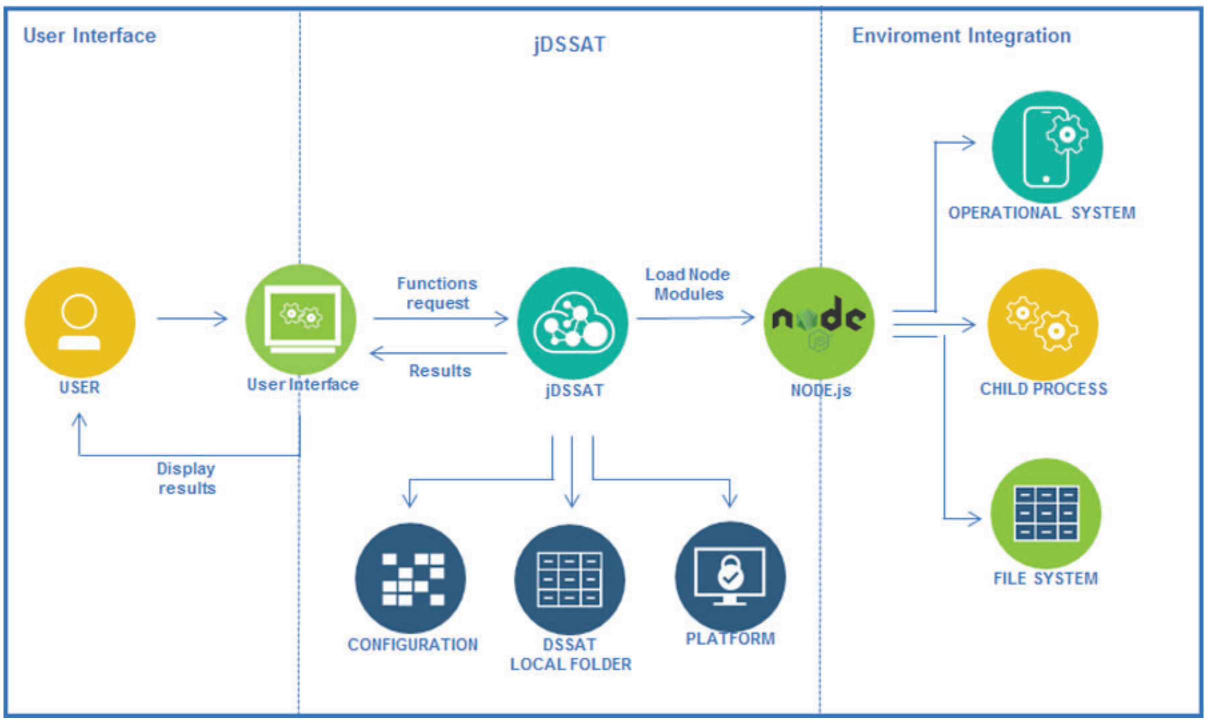


Figure 6. The jDSSAT components architecture and operational system integration. The user interface sends a request to jDSSAT to retrieve and display the information such as crops, experiments and treatments. The node modules are used by jDSSAT to run operations such as process FILEX and run commands in DSSAT-CSM.

First of all we must create an instance of the jDSSAT, as describe in the Section 3.5.4, then it will first identifies if platform is running (Windows, Linux or MacOS) by using the Operational System module that provides an object containing the platform name (win32, darwin or linux). After that, the configurations such as DSSAT CSM executable file name, file system patter and folder name for external tools are loaded. These configurations are stored in a configuration file to avoid code changes in case some path changes in futures DSSAT releases. Finally, jDSSAT finds what is the latest DSSAT version installed on the user's computer and make this version as default to use during the simulations. As a result of jDSSAT initialization, we can now make use of jDSSAT public functions (Tab 3) for reading, processing and writing DSSAT-CSM files. When a function that requires a folder or file content read, jDSSAT makes a file system call using File System (fs) module from Node. The fs module provides an API for interacting with the file system in a manner closely modeled around standard POSIX function [44].

The DSSAT-CSM provides a mechanism to run a simulation model through command line, which is used by jDSSAT. The command line integration in jDSSAT is made by child process module, that enables access to the Operating System functionalities. Once jDSSAT simulation function is called, a command is executed on the operational system. Other components and implementation details are discussed in later sections.

3.5 IMPLEMENTATION AND TESTING

The jDSSAT implementation is divided into two main components, DSSAT files processing and DSSAT-CSM integration. This section discusses the implementation of the current release of jDSSAT and also how we tested its functions.

3.5.1 DSSAT files processing

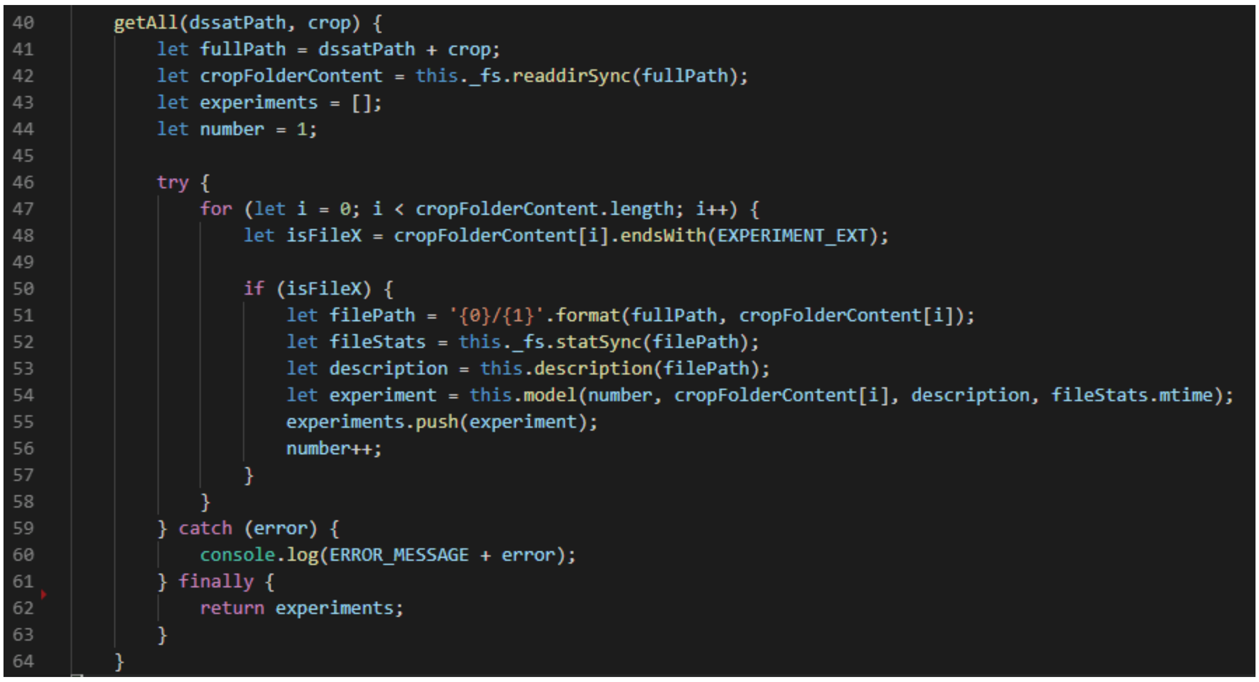
The jDSSAT is able to process FILEX for a specific experiment. This files contains data on treatments, field conditions, crop management and simulation controls [9] and they are found within crop folders with **.*X* extension. Also, jDSSAT can process *OUTPUT* file generated by the crop model that contains results of a simulation. The General DSSAT Profile, named as *DSSATPRO*, is processed to provide information such extension, command line and path for each crop.

3.5.1.1 Crop Folders

The jDSSAT uses DSSAT installation directory content to find *DSSATPRO* file, which designates the locations of all programs and data files used in DSSAT [38]. The DSSATPRO extension file depends on the DSSAT version installed, on Windows, Linux and MacOS, the latest DSSAT version released is **v47**. In other to provide a list of available crops in DSSAT, the jSSAT must read and parse the `DSSATPRO.v47` file, on Windows, to build an array of objects containing crop name, location folder and crop extension. The folders that don't have FILEX in its content will not be returned.

3.5.1.2 Reading Experiments

The read experiment function will search by all FILEX within folder's content. The function will look for files ending with *.*X extension in the crop folder. As seens in the Figure 7, for each FILEX file the algorithm creates an array of experiments with an object containing the description, modified date, name and number as object fields.



```
    getAll(dssatPath, crop) {
        let fullPath = dssatPath + crop;
        let cropFolderContent = this._fs.readdirSync(fullPath);
        let experiments = [];
        let number = 1;

        try {
            for (let i = 0; i < cropFolderContent.length; i++) {
                let isFileX = cropFolderContent[i].endsWith(EXPERIMENT_EXT);

                if (isFileX) {
                    let filePath = '{0}/{1}'.format(fullPath, cropFolderContent[i]);
                    let fileStats = this._fs.statSync(filePath);
                    let description = this.description(filePath);
                    let experiment = this.model(number, cropFolderContent[i], description, fileStats.mtime);
                    experiments.push(experiment);
                    number++;
                }
            }
        } catch (error) {
            console.log(ERROR_MESSAGE + error);
        } finally {
            return experiments;
        }
    }
```

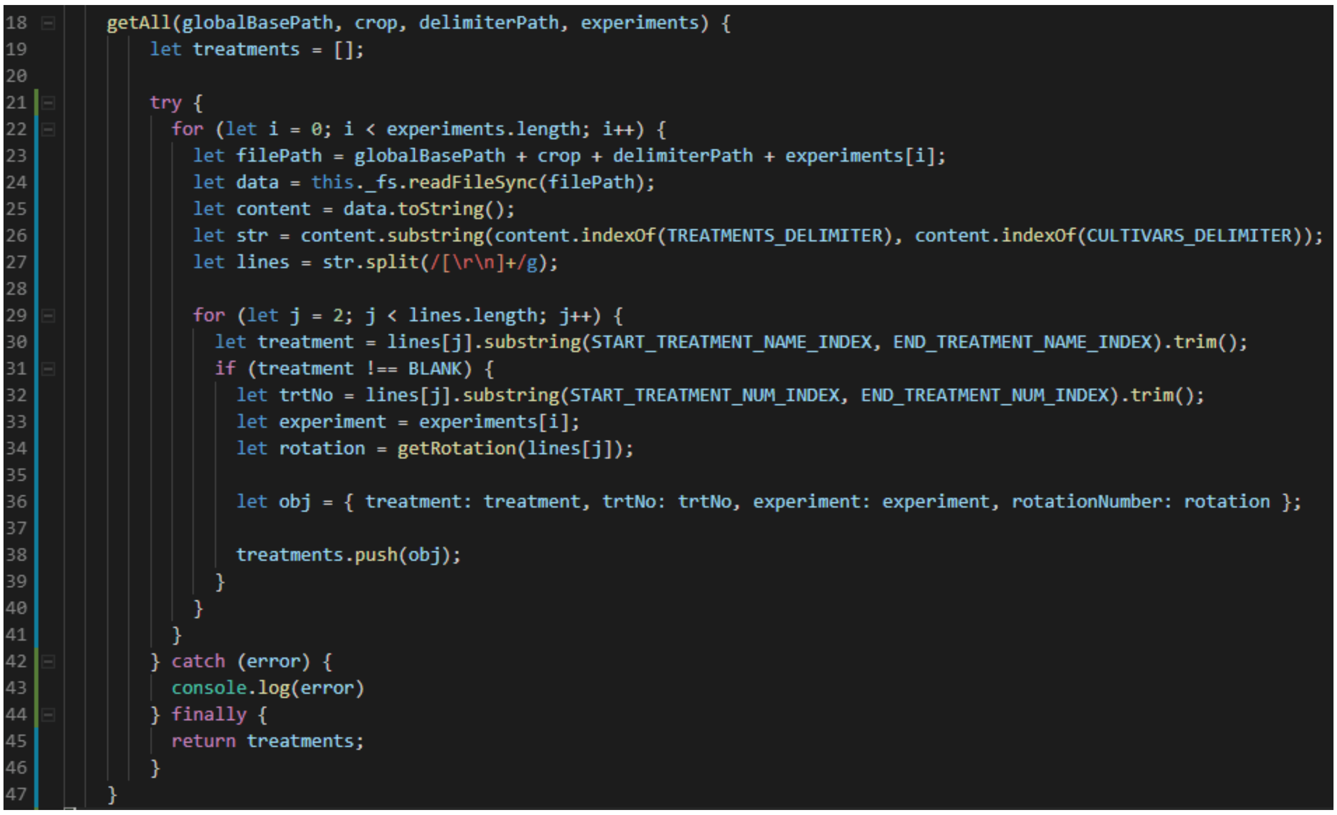
Figure 7. Get all experiments for a given crop functions. The function get a crop as input and finds all FILEX in the crop folder content. The result of the function is an array of experiment objects.

3.5.1.3 Reading Treatments

The treatments function, Figure 8, will receive an array of FILEX names as input. Each FILEX contains a tratments section, as shown in the Figure 9 lines 17 to 22, that will be used to retrieve tratments information. The jDSSAT first loads the FILEX content, then it will parse the content by doing a substring of the index of **TREATMENTS* until the index of **CULTIVARS*. After that, a loop is made through lines of the substring to get the treatment, treatment number and experiment to format an object to send as part of the function response.

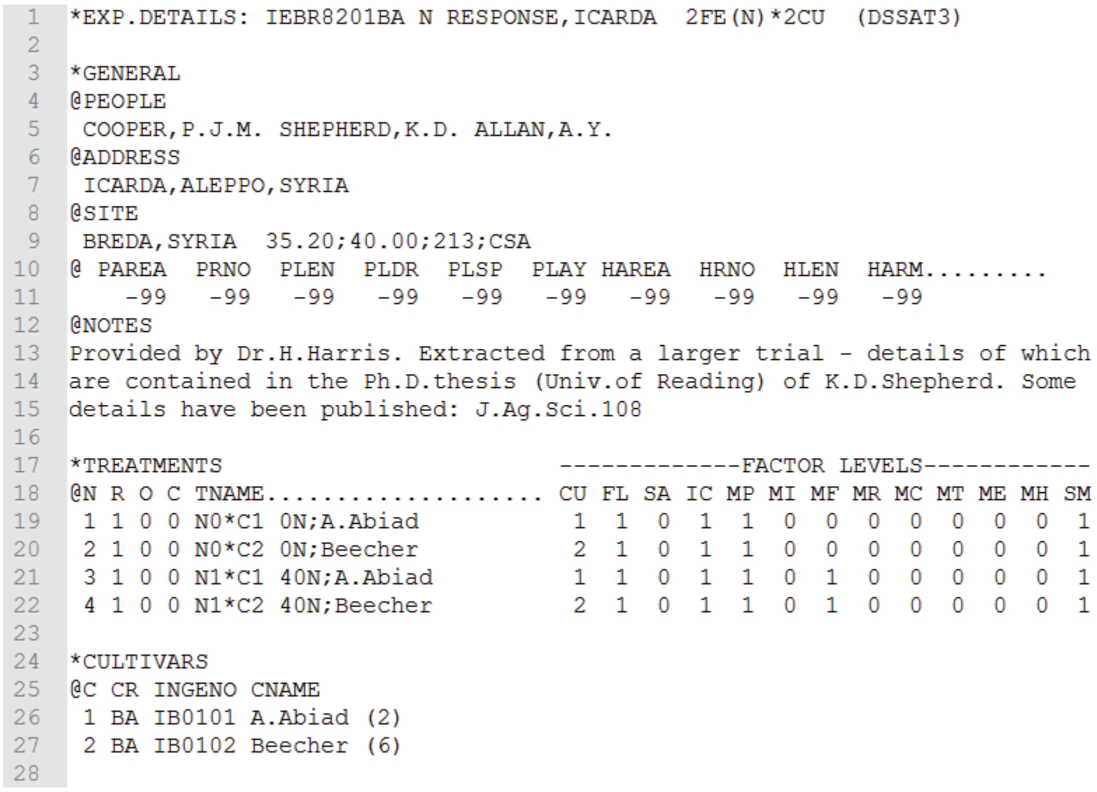
3.5.1.4 Reading .OUT files

Crop simulation models can provide very detailed outputs of the simulated crop. However, analyzing the outputs is challenging. The jDSSAT provides an easy way to read and process these files. The result of a simulation is stored in `.OUT` extension files, these files contains a fixed width format. As seen in Figure 10, the line contains are headers that represent variable names, each row after the header contains the variable values. If a simulation ran with more than one experiment (line 7) or treatment (line 9), this file will have more headers and variable values on its content.



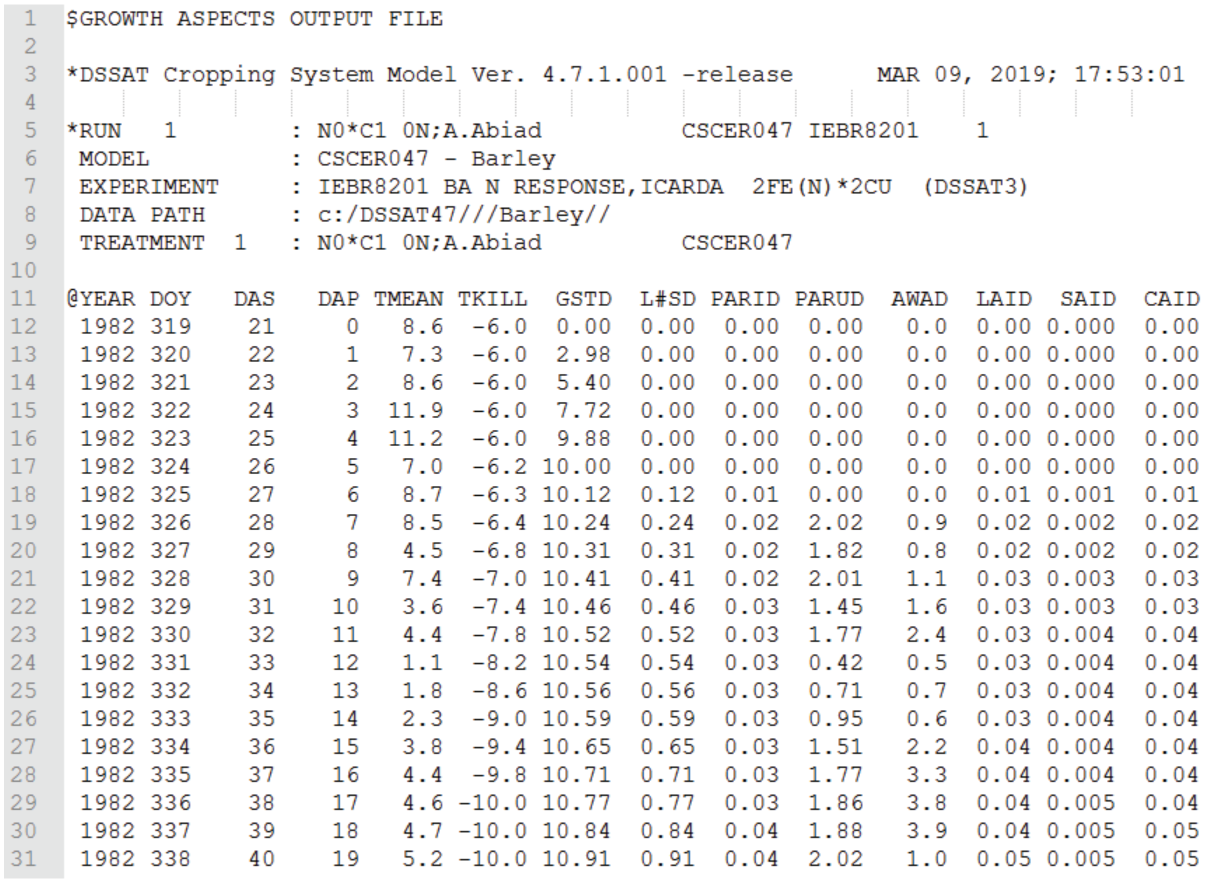
```
getAll(globalBasePath, crop, delimiterPath, experiments) {
    let treatments = [];

    try {
      for (let i = 0; i < experiments.length; i++) {
        let filePath = globalBasePath + crop + delimiterPath + experiments[i];
        let data = this._fs.readFileSync(filePath);
        let content = data.toString();
        let str = content.substring(content.indexOf(TREATMENTS_DELIMITER), content.indexOf(CULTIVARS_DELIMITER));
        let lines = str.split(/[\r\n]+/g);

        for (let j = 2; j < lines.length; j++) {
          let treatment = lines[j].substring(START_TREATMENT_NAME_INDEX, END_TREATMENT_NAME_INDEX).trim();
          if (treatment !== BLANK) {
            let trtNo = lines[j].substring(START_TREATMENT_NUM_INDEX, END_TREATMENT_NUM_INDEX).trim();
            let experiment = experiments[i];
            let rotation = getRotation(lines[j]);

            let obj = { treatment: treatment, trtNo: trtNo, experiment: experiment, rotationNumber: rotation };

            treatments.push(obj);
          }
        }
      }
    } catch (error) {
      console.log(error)
    } finally {
      return treatments;
    }
}
```

Figure 8. Treatment function implementation. For each FILEX received as input the function load its content and finds the tratments section in the file and load a treatments object array as result.



```
 1  *EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)
 2
 3  *GENERAL
 4  @PEOPLE
 5   COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
 6  @ADDRESS
 7   ICARDA,ALEPPO,SYRIA
 8  @SITE
 9   BREDA,SYRIA  35.20;40.00;213;CSA
10  @ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
11      -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
12  @NOTES
13  Provided by Dr.H.Harris. Extracted from a larger trial - details of which
14  are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
15  details have been published: J.Ag.Sci.108
16
17  *TREATMENTS                        -------------FACTOR LEVELS------------
18  @N R O C TNAME.................... CU FL SA IC MP MI MF MR MC MT ME MH SM
19   1 1 0 0 N0*C1 0N;A.Abiad           1  1  0  1  1  0  0  0  0  0  0  0  1
20   2 1 0 0 N0*C2 0N;Beecher           2  1  0  1  1  0  0  0  0  0  0  0  1
21   3 1 0 0 N1*C1 40N;A.Abiad          1  1  0  1  1  0  1  0  0  0  0  0  1
22   4 1 0 0 N1*C2 40N;Beecher          2  1  0  1  1  0  1  0  0  0  0  0  1
23
24  *CULTIVARS
25  @C CR INGENO CNAME
26   1 BA IB0101 A.Abiad (2)
27   2 BA IB0102 Beecher (6)
28
```

Figure 9. Partial content of FILEX (IEBR8201.BAX) from Barley crop. The jDSSAT reads the trat-ments, lines 17 to 22, from this files and returns a formatted response with treatments properties as number and (N) and name (TNAME).



```
 1  $GROWTH ASPECTS OUTPUT FILE
 2
 3  *DSSAT Cropping System Model Ver. 4.7.1.001 -release      MAR 09, 2019; 17:53:01
 4
 5  *RUN   1         : N0*C1 0N;A.Abiad          CSCER047 IEBR8201    1
 6   MODEL           : CSCER047 - Barley
 7   EXPERIMENT      : IEBR8201 BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)
 8   DATA PATH       : c:/DSSAT47///Barley//
 9   TREATMENT  1    : N0*C1 0N;A.Abiad          CSCER047
10
11  @YEAR DOY   DAS   DAP TMEAN TKILL  GSTD  L#SD PARID PARUD  AWAD  LAID  SAID  CAID
12   1982 319    21     0   8.6  -6.0  0.00  0.00  0.00  0.00   0.0  0.00 0.000  0.00
13   1982 320    22     1   7.3  -6.0  2.98  0.00  0.00  0.00   0.0  0.00 0.000  0.00
14   1982 321    23     2   8.6  -6.0  5.40  0.00  0.00  0.00   0.0  0.00 0.000  0.00
15   1982 322    24     3  11.9  -6.0  7.72  0.00  0.00  0.00   0.0  0.00 0.000  0.00
16   1982 323    25     4  11.2  -6.0  9.88  0.00  0.00  0.00   0.0  0.00 0.000  0.00
17   1982 324    26     5   7.0  -6.2 10.00  0.00  0.00  0.00   0.0  0.00 0.000  0.00
18   1982 325    27     6   8.7  -6.3 10.12  0.12  0.01  0.00   0.0  0.01 0.001  0.01
19   1982 326    28     7   8.5  -6.4 10.24  0.24  0.02  2.02   0.9  0.02 0.002  0.02
20   1982 327    29     8   4.5  -6.8 10.31  0.31  0.02  1.82   0.8  0.02 0.002  0.02
21   1982 328    30     9   7.4  -7.0 10.41  0.41  0.02  2.01   1.1  0.03 0.003  0.03
22   1982 329    31    10   3.6  -7.4 10.46  0.46  0.03  1.45   1.6  0.03 0.003  0.03
23   1982 330    32    11   4.4  -7.8 10.52  0.52  0.03  1.77   2.4  0.03 0.004  0.04
24   1982 331    33    12   1.1  -8.2 10.54  0.54  0.03  0.42   0.5  0.03 0.004  0.04
25   1982 332    34    13   1.8  -8.6 10.56  0.56  0.03  0.71   0.7  0.03 0.004  0.04
26   1982 333    35    14   2.3  -9.0 10.59  0.59  0.03  0.95   0.6  0.03 0.004  0.04
27   1982 334    36    15   3.8  -9.4 10.65  0.65  0.03  1.51   2.2  0.04 0.004  0.04
28   1982 335    37    16   4.4  -9.8 10.71  0.71  0.03  1.77   3.3  0.04 0.004  0.04
29   1982 336    38    17   4.6 -10.0 10.77  0.77  0.03  1.86   3.8  0.04 0.005  0.04
30   1982 337    39    18   4.7 -10.0 10.84  0.84  0.04  1.88   3.9  0.04 0.005  0.05
31   1982 338    40    19   5.2 -10.0 10.91  0.91  0.04  2.02   1.0  0.05 0.005  0.05
```

Figure 10. Partial content of PlatGro.OUT file, the read output file function from jDSSAT will process the lines 11 to 31 and as result will return an array of objects with the variable name and its values, so the number of columns in the output file will be the number of objects within the result array.

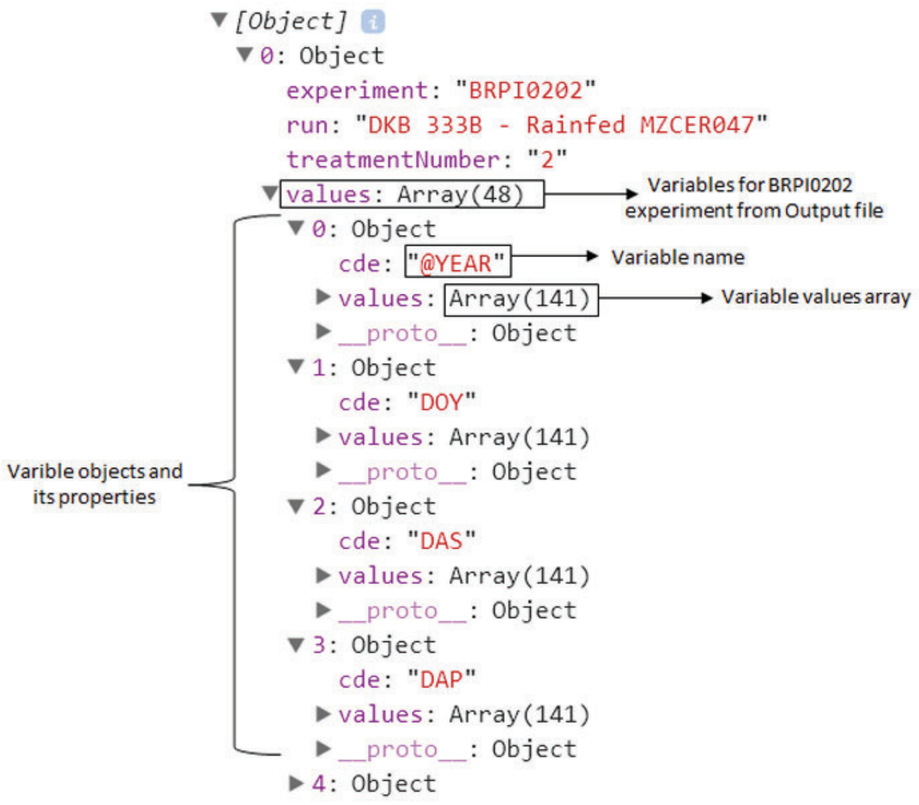
The read output file function receives the crop name and output file name as input. The crop name variable value will be used to identify where the output file is located in the user‘s machine. Once the output file location is found, jDSSAT reads the outfile file content as follows:

1. Initialize a header object with index, lenght, variable name.
2. Initialize an array of objects to store experiment, treatment number, treatment and its values.
3. Loop through the lines.
4. Find what is the run number, experiment and treatment. Each of these has one prefix. This step is performed by looking at RUN, MODEL and EXPERIMENT respectively in the beggining of the line. If an identifier is found, jDSSAT reads the content line that comes after.
5. Read simulation values. The @ symbol followed by YEAR variable is the first character of the data-headers line in DSSAT output files. When jDSSAT finds this symbol in the beginning of the line, it initializes an auxiliary array to store the header. Each value between spaces will be one position on the header array.

6. Get values for each header variable. The algorithm should consider that next lines in the loop will contain the data, similar to *EasyGrapher*' approach [45]. This process should execute while algorithm does not find another run.

7. Repeat all the steps if there are multiple runs in the output file.

The result of reading a DSSAT output file will be an array, we seen in Figure 11, containing an object for each run. The jDSSAT also reads `Summary.OUT` and `Overview.OUT` to provide more information about simulation results.



```
▼ [Object]
  ▼ 0: Object
      experiment: "BRPI0202"
      run: "DKB 333B - Rainfed MZCER047"
      treatmentNumber: "2"
    ▼ values: Array(48)
      ▼ 0: Object
          cde: "@YEAR"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 1: Object
          cde: "DOY"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 2: Object
          cde: "DAS"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▼ 3: Object
          cde: "DAP"
        ▶ values: Array(141)
        ▶ __proto__: Object
      ▶ 4: Object
```

Variables for BRPI0202 experiment from Output file
Variable name
Variable values array
Varible objects and its properties

Figure 11. Read output file function object result. It shows the total variables found and partial variables and its values content.

3.5.2 DSSAT-CSM integration

The DSSAT-CSM incorporates models of all crops within a single set of code. The run mode is specified by the command line arguments when the model is called. If the model is run using the DSSAT shell, these command lines are transparent to the user [9]. There are sets of commands available for CSM, Table 2, the run modes batch and sensitivity are available in the current jDSSAT version. The child process Node module within jDSSAT enables access Operating System functionalities by running any system command.

Run mode	Description	Command line arguments
Batch	Experiments and treatments are listed in a batch file	B batchfilename
Sensitivity analysis	Screen user interface to interactively modify input parameters	E FileX Treatment#
Sequence analysis	Soil processes are continuous, crop sequence is listed in batch file	Q batchfilename
Spatial analysis	Simulates one or more crops over space.	S batchfilename
Seasonal analysis	Multiple years run with the same initial conditions	N batchfilename
Interactive	Screen user interface for interactive selection of experiment and treatment	(none)
Run all treatments	All treatments of specified experiment are run	A FileX
Debug	The input module is not called to read FILEX, soils files or cultivar file. Instead, the temporary input file (inpfile) is read.	D inpfile

Table 2. CSM modes of operation adapted from Overview of DSSAT4 Cropping System Model (CSM) [9].

3.5.2.1 Running a simulation

To run a CSM model first we need to select the experiments and treatments to be simulated. A group of experiments and treatments can be run in a single simulation or in batch mode. A single experiment/treatment can be run either as a batch simulation or in sensitivity mode [17]. The jDSSAT approach is a batch simulation. The simulation is divided into two steps.

1. **Creating a batch file:** The batch file is a set of configurations to run a model such experiments and treatments list. There is a pre-built template of this file within jDSSAT. The function uses this template to fill up the configurations to run a model in batch mode.

2. **Running a command:** After the batch file is created, jDSSAT uses child process node module [46] to execute the command created in the command-line interpreter.

3.5.3 Auxiliary functions

There are also auxiliary functions in other to help users to know additional information, such as the DSSAT version and path being used. The complete list of auxiliary functions can be seen at Table 3.

Function	Description
jdssat.path()	returns the local DSSAT installation path
jdssat.version()	returns the DSSAT version in use by jDSSAT
jdssat.platform()	returns the platform (Win32, Darwin or Linux) where jDSSAT is running
jdssat.tree()	returns an array of folder model objects. The folders will be those that has experiments file inside
jdssat.experimentDescription(experiment, filePath)	returns a string with the experiment description
jdssat.createBashFile(crop, experiments[])	creates a batch file using a template
jdssat.outFiles(crop)	returns an array of output file names
jdssat.cde()	returns an array of objects containing cde, label and description fields
jdssat.openExternalTool(name)	opens a DSSAT Shell external tool
jdssat.openDssatFolder()	opens the current DSSAT folder
jdssat.openFileInEditor(crop,fileName)	opens a file in editor. The editor used depends on the platform
jdssat.getDataFiles(crop)	returns all files in a crop that ending with .*A or .*T extension
jdssat.folders()	returns all folders within DSSAT installation path
jdssat.filePreview(crop, fileName)	returns a file content in html tags
jdssat.runBatchFile(crop)	runs a batch command using child process node module

Table 3. Auxiliary jDSSAT functions and its description

3.5.4 Module testing

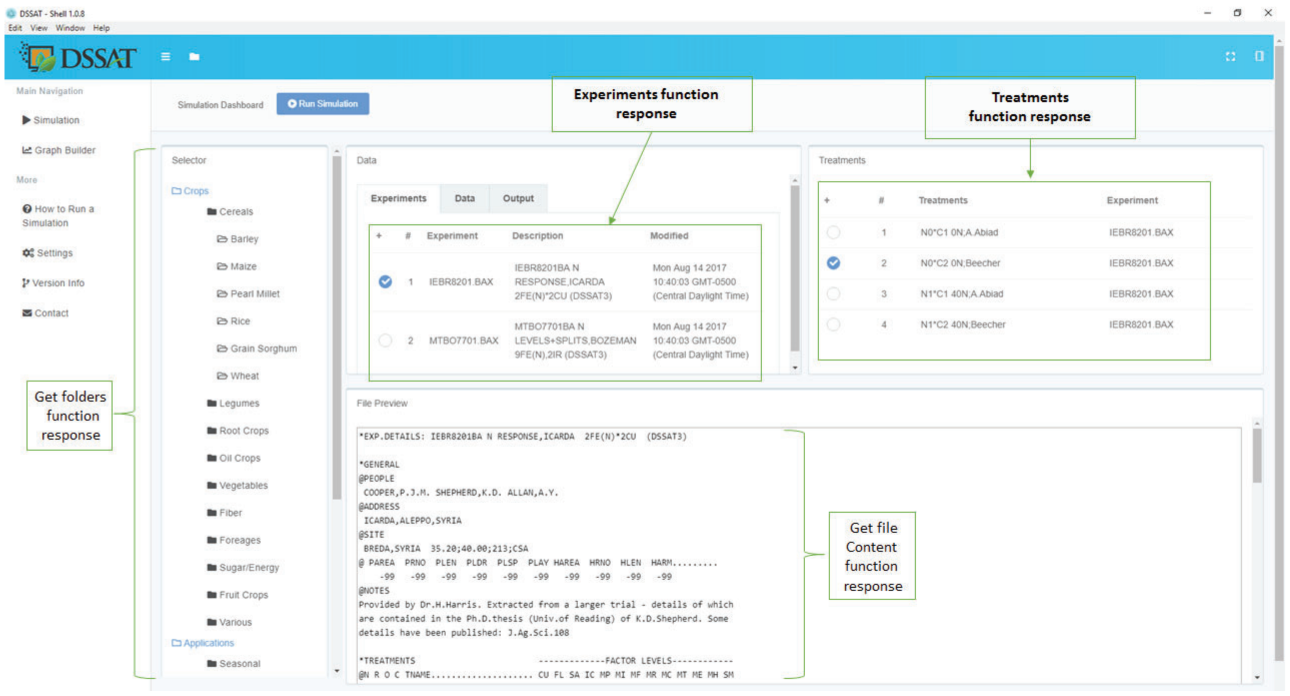
The jDSSAT functions were used to create a user interface, shown in Figure 12, to run DSSAT models. This user interface provides functionalities to the user to choose a crop, experiment, and treatments. Also, it displays a preview of the file that is being processed. The option to run a simulation is available in the blue button on the screen.

A simple R package was created to test the jDSSAT over HTTP. This package download jDSSAT in the local machine and start a API to send request from R to jDSSAT. In the Figure 13 we show a experiment function response in R.

3.6 PRACTICAL USAGE

This section presents the practical usage of jDSSAT and its functions. There is a skeleton project [14] with the basic configuration that needs to be used to run the jDSSAT functions according next sessions' instructions.

[14]https://github.com/jabreuar/electron-skeleton

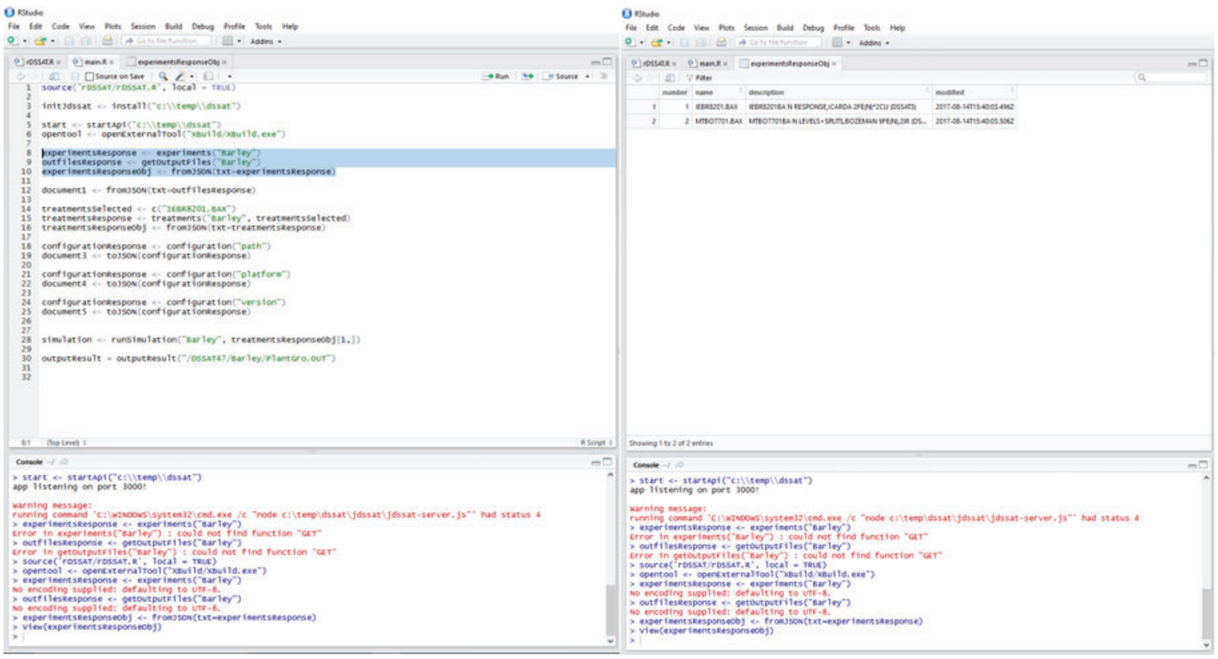


+	#	Experiment	Description	Modified
✓	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	Mon Aug 14 2017 10:40:03 GMT-0500 (Central Daylight Time)
	2	MTBO7701.BAX	MTBO7701BA N LEVELS+SPLITS,BOZEMAN 9FE(N),2IR (DSSAT3)	Mon Aug 14 2017 10:40:03 GMT-0500 (Central Daylight Time)

+	#	Treatments	Experiment
	1	N0*C1 0N;A.Abiad	IEBR8201.BAX
✓	2	N0*C2 0N;Beecher	IEBR8201.BAX
	3	N1*C1 40N;A.Abiad	IEBR8201.BAX
	4	N1*C2 40N;Beecher	IEBR8201.BAX

```
*EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)

*GENERAL
@PEOPLE
 COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
@ADDRESS
 ICARDA,ALEPPO,SYRIA
@SITE
 BREDA,SYRIA  35.20;40.00;213;CSA
@ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
    -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
@NOTES
Provided by Dr.H.Harris. Extracted from a larger trial - details of which
are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
details have been published: J.Ag.Sci.108

*TREATMENTS                        -------------FACTOR LEVELS-----------
@N R O C TNAME.................... CU FL SA IC MP MI MF MR MC MT ME MH SM
```

Figure 12. An user interface to run DSSAT model built using jDSSAT functions responses. The UI has four main components: a) selector, for the user select the crop; b) data, displays the experiments available for the crop select; c) treatments available for the experiment selected; and d) file preview, display FILEX preview.



```
source('rDSSAT/rDSSAT.R', local = TRUE)

initJdssat <- install("c:\\temp\\dssat")

start <- startApi("c:\\temp\\dssat")
opentool <- openExternalTool("XBuild/XBuild.exe")

experimentsResponse <- experiments("Barley")
outfilesResponse <- getOutputFiles("Barley")
experimentsResponseObj <- fromJSON(txt=experimentsResponse)

document1 <- fromJSON(txt=outfilesResponse)

treatmentsSelected <- c("IEBR8201.BAX")
treatmentsResponse <- treatments("Barley", treatmentsSelected)
treatmentsResponseObj <- fromJSON(txt=treatmentsResponse)

configurationResponse <- configuration("path")
document3 <- toJSON(configurationResponse)

configurationResponse <- configuration("platform")
document4 <- toJSON(configurationResponse)

configurationResponse <- configuration("version")
document5 <- toJSON(configurationResponse)


simulation <- runSimulation("Barley", treatmentsResponseObj[1,])

outputResult = outputResult("/DSSAT47/Barley/PlantGro.OUT")
```

```
> start <- startApi("c:\\temp\\dssat")
app listening on port 3000!

Warning message:
running command 'C:\WINDOWS\system32\cmd.exe /c "node c:\temp\dssat\jdssat\jdssat-server.js"' had status 4
> experimentsResponse <- experiments("Barley")
Error in experiments("Barley") : could not find function "GET"
> outfilesResponse <- getOutputFiles("Barley")
Error in getOutputFiles("Barley") : could not find function "GET"
> source('rDSSAT/rDSSAT.R', local = TRUE)
> opentool <- openExternalTool("XBuild/XBuild.exe")
> experimentsResponse <- experiments("Barley")
No encoding supplied: defaulting to UTF-8.
> outfilesResponse <- getOutputFiles("Barley")
No encoding supplied: defaulting to UTF-8.
> experimentsResponseObj <- fromJSON(txt=experimentsResponse)
> View(experimentsResponseObj)
>
```

	number	name	description	modified
1	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	2017-08-14T15:40:03.496Z
2	2	MTBO7701.BAX	MTBO7701BA N LEVELS+SPLITS,BOZEMAN 9FE(N),2IR (DS...	2017-08-14T15:40:03.506Z

```
> start <- startApi("c:\\temp\\dssat")
app listening on port 3000!

Warning message:
running command 'C:\WINDOWS\system32\cmd.exe /c "node c:\temp\dssat\jdssat\jdssat-server.js"' had status 4
> experimentsResponse <- experiments("Barley")
Error in experiments("Barley") : could not find function "GET"
> outfilesResponse <- getOutputFiles("Barley")
Error in getOutputFiles("Barley") : could not find function "GET"
> source('rDSSAT/rDSSAT.R', local = TRUE)
> opentool <- openExternalTool("XBuild/XBuild.exe")
> experimentsResponse <- experiments("Barley")
No encoding supplied: defaulting to UTF-8.
> outfilesResponse <- getOutputFiles("Barley")
No encoding supplied: defaulting to UTF-8.
> experimentsResponseObj <- fromJSON(txt=experimentsResponse)
> View(experimentsResponseObj)
>
```

Figure 13. Experiment function running in R through jDSSAT API. The right side shows a code sample to retrieve the experiments from Barley crop and the left side shows a frame if the result of the experiments function.

3.6.1 Installation

The jDSSAT package [15] is available through Node package manager (npm), the package manager for JavaScript [47]. The download can be done by running *npm install jdssat* com-

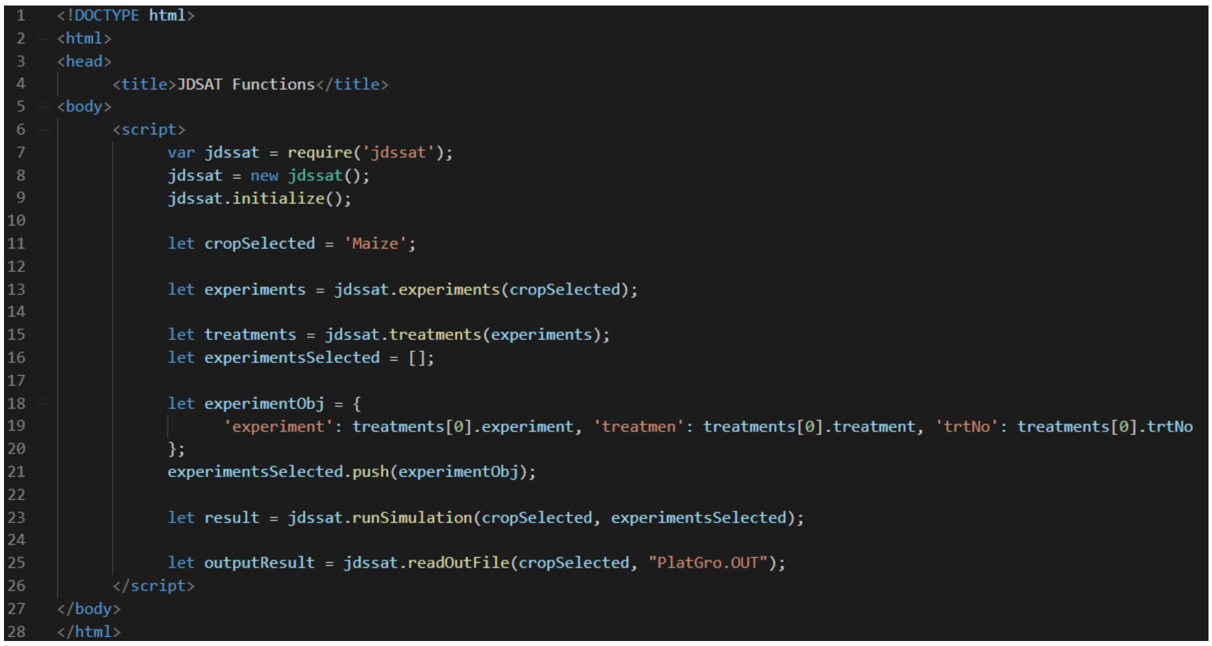
[15]https://www.npmjs.com/package/jdssat

mand. Once the jDSSAT download is completed, a folder is automatically created under node modules.

```
project
├── app
└── node_module
    └── jdssat
        └── source
```

3.6.2 Functions

The code seen in the Figure 14 shows how to initialize and use experiments, treatments, simulation and read output files function. The initialize function requires jdssat module, the default initialization function identifies the platform windows (win32), linux and MacOS(darwin) then it finds the latest DSSAT version installed on user's machine by looking at `"C:/"` on Win32 or `"/"` on Darwin and Linux platform. Also, it loads a series of configuration, such base path for the operating system, the version of DSSATPRO [8] and other extra data for managing the fixed width format properties. The experiments function input is a crop, it reads all FILEX from the crop selected. Also, the experiments are used as input for the treatments function. To run a DSSAT model simulation, the jDSSAT run simulation function expects the crop and an array of objects with the experiments selected. As seen in the Figure 14 (lines 18, 19 and 20), the objects within the array are composed by experiment, treatment and treatment number. Finally, the read output function input is a crop selected and output file name.



```
1   <!DOCTYPE html>
2   <html>
3   <head>
4           <title>JDSAT Functions</title>
5   <body>
6           <script>
7                   var jdssat = require('jdssat');
8                   jdssat = new jdssat();
9                   jdssat.initialize();
10
11                  let cropSelected = 'Maize';
12
13                  let experiments = jdssat.experiments(cropSelected);
14
15                  let treatments = jdssat.treatments(experiments);
16                  let experimentsSelected = [];
17
18                  let experimentObj = {
19                          'experiment': treatments[0].experiment, 'treatmen': treatments[0].treatment, 'trtNo': treatments[0].trtNo
20                  };
21                  experimentsSelected.push(experimentObj);
22
23                  let result = jdssat.runSimulation(cropSelected, experimentsSelected);
24
25                  let outputResult = jdssat.readOutFile(cropSelected, "PlatGro.OUT");
26          </script>
27  </body>
28  </html>
```

Figure 14. Sample of jDSSAT functions to initialiaze, read experiments and treatments, run simulation and retrieve results.

3.7 FINAL CONSIDERATIONS

In this chapter, we have presented jDSSAT, a flexible JavaScript module that abstracts the complexity of reading and processing DSSAT files. The module integrates with DSSAT-CSM in different operational system for running models. The implementation of jDSSAT also highlights interoperability benefits, which have been an issue in the DSSAT tools. Also, the jDSSAT is designed to be a standard module for DSSAT integration which reduces the effort to build integrations with DSSAT in other programming languages such as R.

The jDSSAT is an ongoing project. There is an oportunity to to reuse its code to have a version to run DSSAT models on the web that would allow run DSSAT as a service. Also, the jDSSAT can be used to develop an end-to-end user interface as an alternative for the current DSSAT Shell.

4. DSSAT-LITE: A WEB-BASED APPLICATION FOR RUNNING CROP MODELS AND ANALYZING RESULTS FROM DSSAT-CSM

4.1 ABSTRACT

DSSAT is a set of on-premises tools that facilitates the creation and the management of experiments files, soil and climate. The DSSAT tools were built using software programming languages such as R, Visual Basic and Delphi which become hard to run remotely or in a cloud farm. The installation of the software on the user's machine is not recommended nowadays because managing release version and support different environments is a big challenge for software development teams. The web development has been changing how software is built and allowing many others functionalities such as parallelism, accessibility, cross-platform and scalability. In this work, we present DSSAT-Lite, a web-based application designed to facilitate DSSAT-CSM models simulation over HTTP. The DSSAT-Lite provides a simulation and visualization tools and smooth integration with highly optimized third party software for DSSAT files processing. Abstraction layers enable a standardized methodology to read various DSSAT files into a very well formated data structure. These features have allowed DSSAT-Lite to aid remote simulations specifically using Mobile devices and web browsers.

4.2 MOTIVATION AND SIGNIFICANCE

The web has existed for more than two decades and been used for many purposes, including as a source of information, entertainment, and much else. It has become an indispensable part of our daily lives [48].

In the last few years, several web technology innovations have allowed software designers and engineers to quickly develop responsive web applications [49]. The web-based approach also reduces the complexity of software distribution, which is a challenge for any development team.

The bursting of the dot-com bubble in 2001, marked a turning point for the web [50]. Many tools were built before that, using technologies that become hard to maintain nowadays.

The tool that drives this work is the Decision Support System for Agrotechnology Transfer (DSSAT), a software package that ingerates soil effects, genetic coefficients, climate data and management options. DSSAT is a powerfull tool for researchers and policy-makers for decision-making and to answer *what if* questions related to a cropping system [51]. Most of the operations in DSSAT are made over command-line interpreter, which requires a good computational knowledge. Since the DSSAT community are interdisciplinary, not only computer science or others are related, the use of command-line become a challenge for most of the users. Many times, it is a challenge for researchers on the computer science area as well.

There is one alternative for command-line, which is the DSSAT Shell program. It provides a working environment in which various stand-alone tools and applications are seamlessly integrated with the DSSAT crop models. Within the shell, the user can launch applications for creating and modifying data files, running the crop models, and analyzing the results [52].

However, the current DSSAT Shell is a desktop application that runs on Windows Operation System only. As any other desktop application, it requires to install in a local machine and simulation and data visualization are accessed through offline mode.

This study will help the whole DSSAT comunity, more than 14,000 researchers, educators, consultants, extension agents, growers and decision makers in over 150 countries worldwide [53]. The goal of this work DSSAT Shell is built a solution that allow DSSAT comunity to run and visualize DSSAT crop models over HTTP, to run simulations and visualize results from anywhere in the globe, through mobile device or via browser using World Wide Web.

The remainder of this chapter is structured as follows. Section 2 presents the key technical requirements of DSSAT-Lite application development. Section 3 presents the technical implementation details, design and architecture. Section 4 contains illustrated examples of the DSSAT-Lite main features and their implementation details. Finally, Section 5 presents our conclusions and directions for future research.

4.3 SOFTWARE DESCRIPTION

In this chapter, we describe the functional requirements and functionalities that have guided our implementation. We also present what are the main functionalities from DSSAT Shell that will be supported by DSSAT-Lite.

4.3.1 Functional requirements

- Responsive: DSSAT-Lite pages should render well on a variety of devices and window or screen sizes. It also must support mobile devices;
- Parallelism: it must support multiple simulation at once. Simulations should not impact each other;
- Performance: slow sites have a negative impact for the users. DSSAT should be able to run simulations and visualize result quickly;
- User-friendly: DSSAT-Lite must provide an user-friendly interface to the users; even users that are starting with DSSAT should be able to find the path to run a simulation and also to visualize results easily;

4.3.2 Software functionalities

As shown in Figure 16, there are four menus on the left, Simulatition, Graph Builder, Version Info and Contact. Simulatition contains information about the experiments and treatments. Graph Builder is used to plot graph to visualize simulation results. The other two menus are version info which contains the release note of the current version and contact that is used to report issues. At the top of the Simulation page, there is a dropdown with the available crops to be selected. This means that the users can select differnt crop to run asimulation. Next, the button Run Simulation is used to run a simulation in the remote server after Experiments and treatments are select.

4.4 TECHNICAL IMPLEMENTATION

In this section, we review some concepts and usage of the JavaScript frameworks as well as the architecture used during the DSSAT-Lite development.

4.4.1 Framework and Libraries

Many frameworks and libraries were used during DSSAT-Lite development. We review these frameworks in the next subsections.

4.4.1.1 jDSSAT

The jDSSAT is a JavaScript library, which in publishing process, was created to read, write and process DSSAT-CSM files. It uses DSSAT data standards [54]. The library is able to integrate with diferente output files such as Plant growth and Soil carbon.

jSSAT provide a set of functions to retrive experiments and treatments. It also is capable to run DSSAT-CSM model simulations and also process the results.

This library provides a Node.js server abstraction. It works on either client side or server side. On this work, we used jDSSAT on the server side because it will handle the HTTP request from the client as an Application Program Interface (API).

4.4.1.2 Express.js

Express.js, is a web application framework for Node.js. It is designed for building web applications and APIs [55]. Node.js couple JavaScript with an event loop for quickly dispatching operations when events occurs. Node.js's philosophy is to give low-level access to the event loop and system resources [46]. On this work we use Express to build a middleware API. It handles requests from a client to read, write or process DSSAT-CSM files.

Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application's request-response cycle. Once a request function is invoked, it executes functions from jDSSAT, section 4.4.1.1, to handle the request. These functions can, for example, be read by experiments from an experiment file or run a simulation. The API is hosted in a virtual machine, it handles HTTP requests and ask the response to the jDSSAT (integration layer with DSSAT files). More details are discussed on the Section 4.4.2.

4.4.1.3 Angular

Angular is a TypeScript-based open-source front-end web application platform led by the Angular Team at Google [56, 57]. Angular applications use the component patter [58]. It involves combinig smaller, discrete building blocks into larger finished products. For instance, a battery ia a component of a an automobile.

Angular is used to develop the DSSAT-Lite front-end (what users see). The use of Angular allow us to give the visibility of the steps to run a DSSAT simulation into components. For example, select a crop, experiments and treatments are isolated components.

4.4.1.4 Bootstrap

Responsive web design adds more layers of complexity to design and develop websites [59]. In this sense, we have choosen Bootstrap: it is a free and open-source front-end framework for designing websites. Our goal in terms of layout is make it as much cleaner and intuitive as possible.

4.4.1.5 Jasmine

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks [60]. The use of Jasmine allow us to cover our code with tests. This technique helps any developer making changes and make sure nothing has broken. It also helps understanding a software flow.

4.4.2 Architecture Review

The DSSAT-Lite application design is based on an architecture that follows the Client/Server approach via Representational State Transfer (REST) [61].All the components of the architecture are shown in the Figure 15.

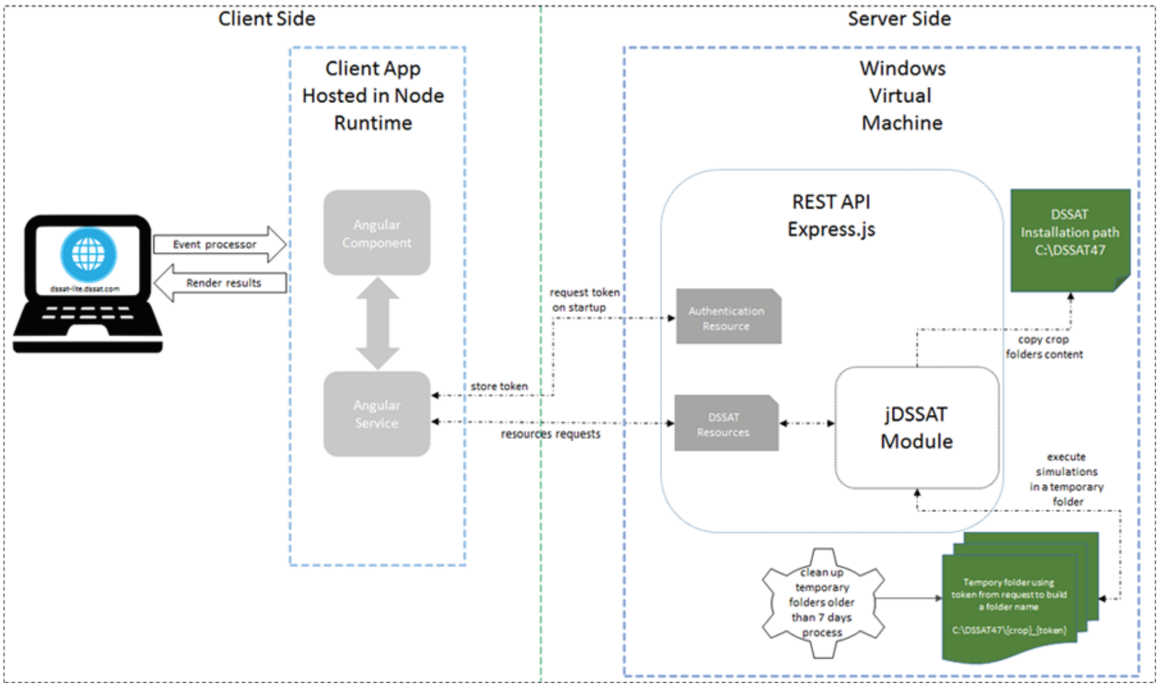

Client Side
Server Side
Client App Hosted in Node Runtime
Windows Virtual Machine
dssat-lite.dssat.com
Event processor
Render results
Angular Component
Angular Service
request token on startup
store token
resources requests
REST API Express.js
Authentication Resource
DSSAT Resources
jDSSAT Module
DSSAT Installation path C:\DSSAT47
copy crop folders content
execute simulations in a temporary folder
clean up temporary folders older than 7 days process
Tempory folder using token from request to build a folder name
C:\DSSAT47\{crop}_{token}

Figure 15. DSSAT-Lite Architecture. The client side make HTTP calls to the server, where DSSAT is installed. The server handles the request and returns the simulation results to the client.

Once the user access the DSSAT-Lite Uniform Resource Locator (URL), a request is made to the Middleware REST API, specifically to the authentication service, in order to generate a token. We keep the token in the browser local storage [62]. This token will be used later on to create a temporary crop folders to avoid concurrence issues. The simulation from user *A* won't impact a simulation from user *B*, both users will have different temporary folders.

After the token is generated and stored, we can now use DSSAT resource available in the Middleware API. The DSSAT resource has a different path in the URL, for instance, to retrieve experiments for a given crop, the URL [16] path will contains the crop name. The HTTP method on this case will be a GET. The result of this request will be a JSON Array [63] and it will be handle by the client side before displaying it to the user.

DSSAT-Lite supports any user to run a DSSAT-CSM simulation model remotely. In this case, a temporary folder with the results will be created. For example, the user *A* acccess the DSSAT-Lite URL, at this moment the following token *9f5IEN7BWP* is generate by the middleware API. The user then will select what: the crop, experiments and treatments to be simulated, on this

[16]https://fqdn/api/dssat/treatments:crop

order, and push *Run Model* button in the DSSAT-Lite page, a request is made to the Middleware API to run the simulation. This operation is performed as follows:

1. Create a temporary crop folder. The folder name will contains the crop name used during the simulation, such as Maize, Barley, and others. Taking Maize as DSSAT-CSM crop example, the folder name would be *C:\DSSAT47\Maize_9f5IEN7BWP*.
2. Run Simulation: After a temporary crop folder, to store the Batch and Simulation results, is created, jDSSAT will run the simulation using that folder. The bash file created, in order to run the simulation, will use the temporary folder as well. After the simulation is finished the Middleware API returns the simulation status to the client.

4.5 ILLUSTRATIVE EXAMPLES

DSSAT-Lite has been implemented over HTTP with the integration of ploty.js library [64] for graphical results. Furthermore, this solution enables multiple simulations simultaneously using APIs [17].

The first case, depicted in Figure 16, represents the Maize crop selection by the DSSAT-Lite user. Whenever an event occur in the Web-interface a request is made to the server where DSSAT-Lite backend is hosted. Once the server receives the request, it looks by experiments files available from Maize crop folder. The treatments available for selection are based one the experiment selected. After tretments and experiments are selected the user can now run a simulation. A graphical representation of running simulation steps is shown in the Figure 17.

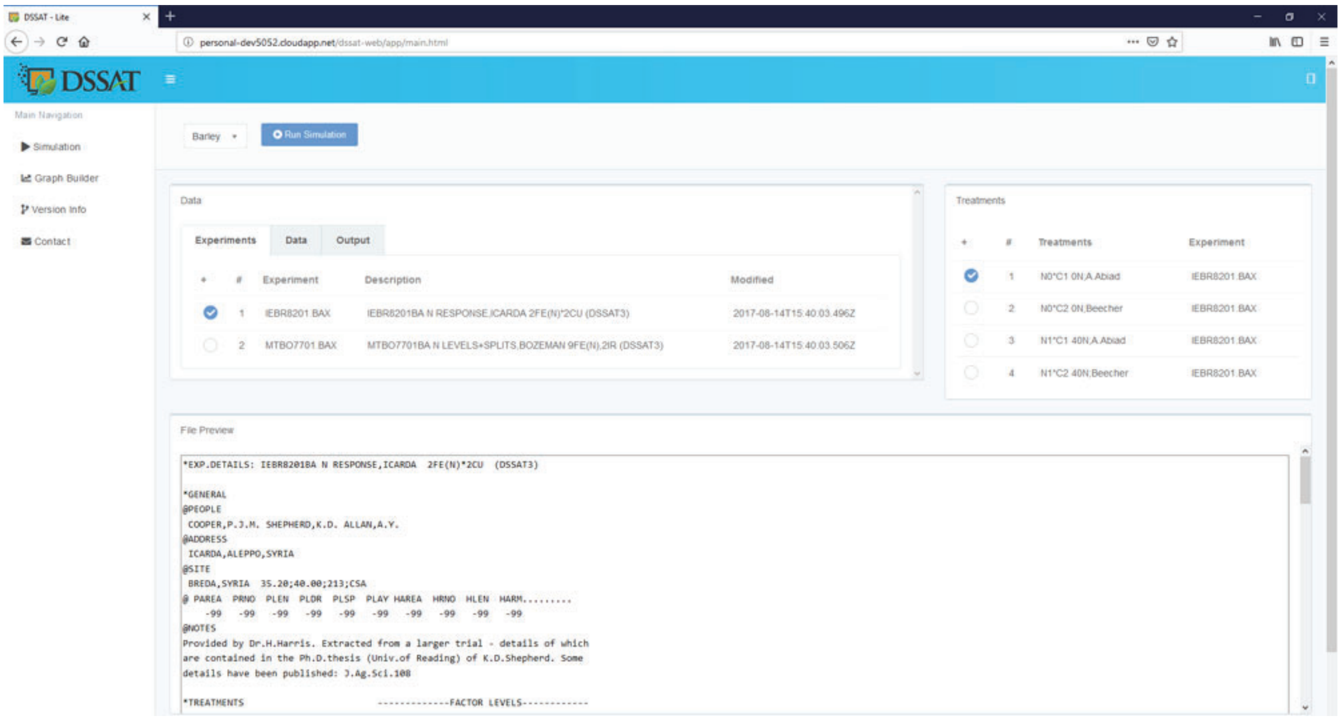
After a simulation is completed the user can visualize the results. Crop simulation models are process-based and provide very detailed outputs of the simulated crop and soil processes [65]. However, analyzing the outputs is challenge and the existing tools such Easy Grapher [65] do not run through web. The DSSAT-Lite provides a page to visualize simulation results, shown in the Figure18.

The DSSAT-Lite is responsive, it automatically resize, hide, shrink, or enlarge its content, to make its user interface fits in all devices (desktops, tablets, and phones). The Figure 19, shows how the DSSAT-Lite layout fits in a mobile device.

4.6 FINAL CONSIDERATIONS

The philosophy of abstracting complexity from software execution should enable the evolution of DSSAT as a software. DSSAT-Lite combines APIs and third-party tools to allow the DSSAT community to run experiments remotely. In the current DSSAT-Lite version, the users

[17]application programming interface



	#	Experiment	Description	Modified
✓	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	2017-08-14T15:40:03.496Z
	2	MTBO7701.BAX	MTBO7701BA N LEVELS+SPLITS,BOZEMAN 9FE(N),2IR (DSSAT3)	2017-08-14T15:40:03.506Z

	#	Treatments	Experiment
✓	1	N0*C1 0N;A.Abiad	IEBR8201.BAX
	2	N0*C2 0N;Beecher	IEBR8201.BAX
	3	N1*C1 40N;A.Abiad	IEBR8201.BAX
	4	N1*C2 40N;Beecher	IEBR8201.BAX

```
*EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)

*GENERAL
@PEOPLE
 COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
@ADDRESS
 ICARDA,ALEPPO,SYRIA
@SITE
 BREDA,SYRIA  35.20;40.00;213;CSA
@ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
    -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
@NOTES
Provided by Dr.H.Harris. Extracted from a larger trial - details of which
are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
details have been published: J.Ag.Sci.108

*TREATMENTS                        ------------FACTOR LEVELS-----------
```

Figure 16. DSSAT-Lite simulation page. The data card displays all experiments based on crop selected, Maize on this case. The treatments displayed on the right card are loaded based on the experiment file the user selected. There is also a File Preview card of the experiment file

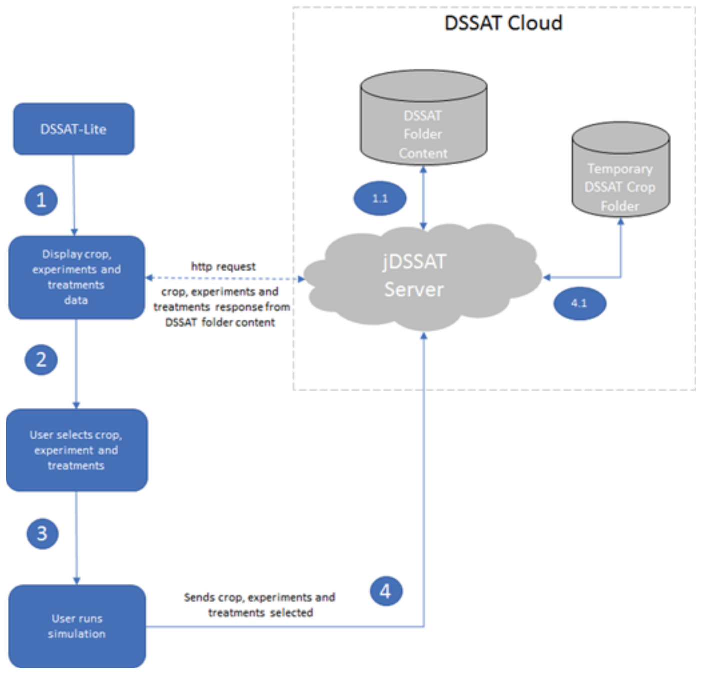

DSSAT Cloud
DSSAT-Lite
DSSAT Folder Content
Temporary DSSAT Crop Folder
1
1.1
Display crop, experiments and treatments data
http request
crop, experiments and treatments response from DSSAT folder content
jDSSAT Server
4.1
2
User selects crop, experiment and treatments
3
User runs simulation
Sends crop, experiments and treatments selected
4

Figure 17. Flow diagram showing the steps to run a simulation. The step 1 is for the DSSAT-Lite load the crops, experiments ad treatments using jDSSAT server API (step 1.1). Step 2 is represents DSSAT-Lite displaying the data. The step 3 represents experiment and treatments selection by the user. Finally the user runs a simulation, there is also a step 4.1 that is for jDSSAT server to process the simulation request and create a temporary folder to save the simulation results and retrieve later on.

encounter limited features in usage when comparing with current DSSAT-Shell. However, running application through the web opens up the possibility to run multiple simulations at once.

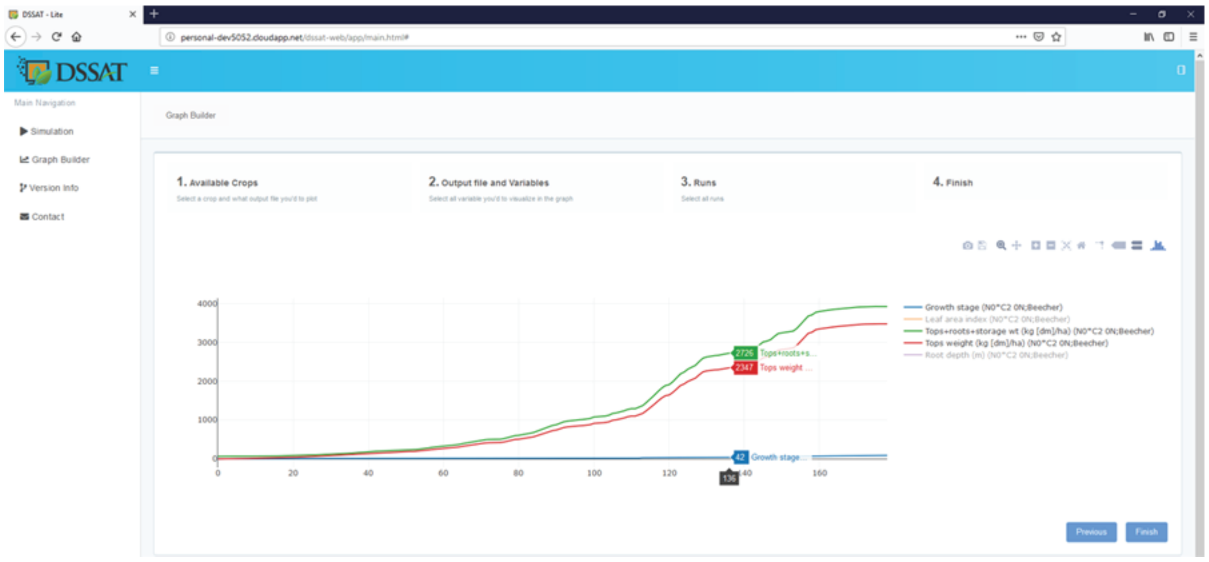
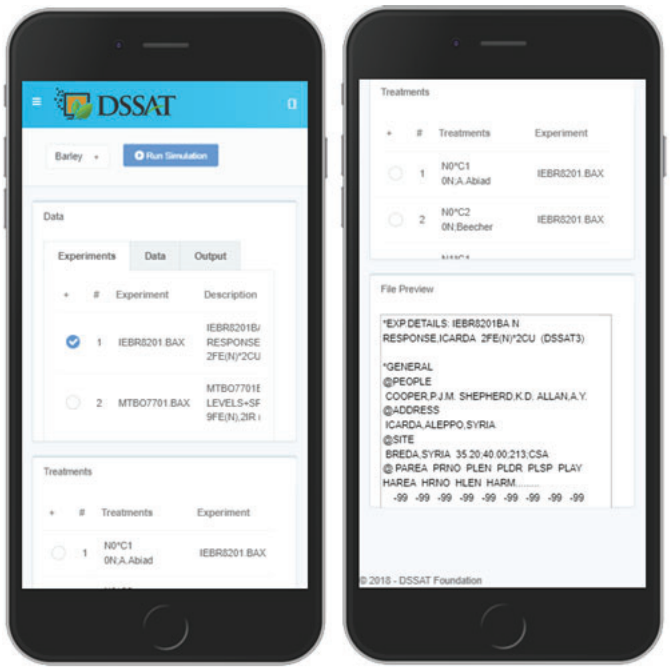

DSSAT - Lite
personal-dev5052.cloudapp.net/dssat-web/app/main.html#
DSSAT
Main Navigation
Simulation
Graph Builder
Version Info
Contact
Graph Builder
1. Available Crops
Select a crop and what output file you'd to plot
2. Output file and Variables
Select all variable you'd to visualize in the graph
3. Runs
Select all runs
4. Finish
Growth stage (N0*C2 0N;Beecher)
Leaf area index (N0*C2 0N;Beecher)
Tops+roots+storage wt (kg [dm]/ha) (N0*C2 0N;Beecher)
Tops weight (kg [dm]/ha) (N0*C2 0N;Beecher)
Root depth (m) (N0*C2 0N;Beecher)
2726 Tops+roots+s...
2347 Tops weight ...
42 Growth stage ...
136
Previous
Finish

Figure 18. Graph builder page. It display the variables values from a simulation. It has options to download the graph, zoom in and out and show/hide variables from the graph.



+	#	Experiment	Description
✓	1	IEBR8201.BAX	IEBR8201B/ RESPONSE 2FE(N)*2CU
	2	MTBO7701.BAX	MTBO7701E LEVELS+SF 9FE(N),2IR

Treatments

+	#	Treatments	Experiment
	1	N0*C1 0N;A.Abiad	IEBR8201.BAX
	2	N0*C2 0N;Beecher	IEBR8201.BAX

File Preview

```
*EXP.DETAILS: IEBR8201BA N
RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)

*GENERAL
@PEOPLE
 COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
@ADDRESS
 ICARDA,ALEPPO,SYRIA
@SITE
 BREDA,SYRIA  35.20;40.00;213;CSA
@ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY
HAREA  HRNO  HLEN  HARM.........
   -99  -99  -99  -99  -99  -99  -99  -99  -99
```

© 2018 - DSSAT Foundation

Figure 19. DSSAT-Lite running in a mobile device (iPhone 6/7/8 plus).

The APIs used by DSSAT-Lite are versatile and extensible. These APIs can be used to built other solutions, such as R package, to run simulations using R language; leveraging to its power of data visualization.

5. DSSAT-SHELL FOR DESKTOP: A CROSS-PLATFORM

5.1 ABSTRACT

The DSSAT is a set of computer programs to simulate the growth of agricultural crops. It has been used in over 100 countries by agronomists to evaluate agricultural methods. The DSSAT-CSM incorporates models of all crops within a single set of code. The run mode is specified by the command line arguments when the model is called. However, if the model is run using the DSSAT shell, these command lines are transparent to the user. The DSSAT shell was originally developed for a Windows environment and DSSAT users from other systems such as Linux and Mac needs to deal with command line arguments to execute the models. Also, others DSSAT stand-alone programs were built to run in windows only and using outdated technologies. In this work, we present a cross-platform application built to work in different operating systems such as Windows, Linux or Mac for running the DSSAT models. We also present the benefities of the cross platform development approach as well as stadard designed to facilitate the development of new DSSAT tools using cross-platform techniques. The interface allows users to run simulation and visualize its result from Windows, Linux and MacOS. It includes many enhancements over the current DSSAT-Shell such as distribution and more frendly user interface, providing a GUI toolkit for efficient interactive work.

5.2 MOTIVATION AND SIGNIFICANCE

The decision support system for agrotechnology transfer (DSSAT) was originally developed by an international network of scientists, cooperating in the International Benchmark Sites Network for Agrotechnology Transfer project, to facilitate the application of crop models in a systems approach to agronomic research [8]. DSSAT are a collection of databases, models and different applications that are operated by software to help the user select and compare different options and predict the final results [15]. DSSAT has been in used by more than 14,000 researchers, educators, consultants, extension agents, growers, and policy and decision makers in over 150 countries worldwide [66]. It has significant importance as a really useful tool for researchers and policy-makers for decision-making and to answer what – if questions related to a cropping system [8].

The models in DSSAT are executed through the command-line interpreter that requires the user a certain experience in computer science. There is a DSSAT-Shell, Figure 20, that includes many enhancements over the terminal command line execution. However, this Shell runs on Windows only and it was written using Visual Basic language. Visual Basic is a thirdgeneration event-driven programming language from Microsoft for its Component Object Model programming model [67] first released in 1991 and declared legacy during 2008.

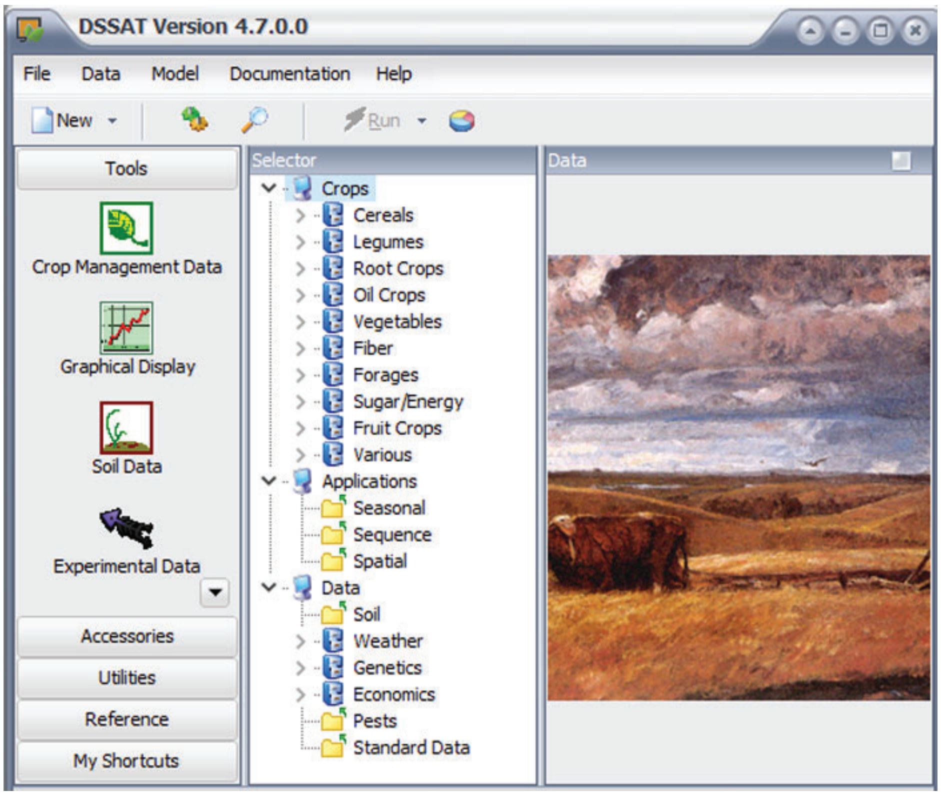


Figure 20. Screenshot from the main DSSAT-Shell window, available under DSSAT version 4.7 installation package.

One of the biggest problems with the current DSSAT-Shell is that Visual Basic is at the end of life support. The Visual Basic 6 runtime is part of Windows 8 and therefore will be supported in the same way as the rest of Windows [68]; for example, Windows 8 operating system will have its support until 2024, as Visual Basic follows Windows 8 support lifecycle it will be decommissioned soon.

In order to remove technology limitations and keep DSSAT-Shell supportable, we developed a new DSSAT-Shell, with basic functionalities. The new shell includes many enhancements over the existing shell, including the capability of runnig on different operational systems such as Windows, Linux and MacOS, as well as interactive graphical results. More DSSAT tools will be implemented continuously using the same technologies.

5.3 APPROACHES FOR CROSSPLATFORM APPLICATION

Platforms are a combination of hardware, operating system, SDKs and standard libraries, which together offer the base for building software for that platform [69]. As all platforms differ significantly from each other, software developers that want to reach a large audience of users would be required to develop their apps for each platform separately [70].

An application can be made cross-platform by combining different methods. These methods can achieve this in application building phase, while other may achieve it in application execution phase [71]. Many cross-platform frameworks can help developers to make their software cross-platform compatible. The cross-platform frameworks allows reusing parts of the application source code for multiple platforms and may additionally offer building blocks such as user interface API or other functionality that is not specific to a single platform.

The main idea behind developing cross-platform desktop applications using JavaScript is that there a single codebase and it is packaged for each operating system separately. This abstracts the knowledge needed to build native desktop applications and makes maintenance easier.

There are diffent cross-plataform approaches that can be classified primarily into following categories [72]:

- Web approach: browser-based applications;
- Hybrid approach: combines web and native technologies;
- Interpreted approach: automatically code generations to implement user interfaces;
- Cross compiled or generated approach: compiled apps just like a native app;

Nowadays, developing a desktop application using web technologies relies on frameworks such as Electron [73]. It is a blend of two popular technologies: Node.js and Chromium. Thus, any web application can run on Electron as well as Node application can run on Electron [74]. Electron takes the advantage of Chromium, an open-source browser project that aims to build a safer, faster, and more stable way for users to experience the web [29].

Electron-based applications run in two distinct processes: the main process and the render process, Figure 21.

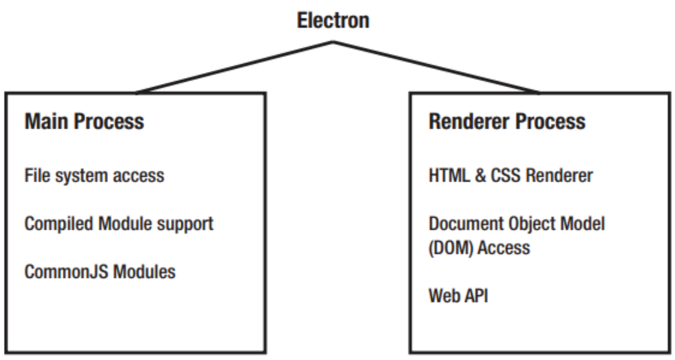


Figure 21. The mais process contains all modules to integrate with operational system such as file system. It also loads common javascript modules such as path, that abstracts complexity for reading folder content. The renderer process is responsible to render the html and css code, it also provices access to the document objecto model.

Within the main process is where the application will handle various system-level activities, like life-cycle events (starting up, preparing to quit, actually quitting, switching between the foreground and background, as just a few examples). The main process also has another responsibility, which is to spawn any render processes. These processes will display the UI of the application. Each of these render processes will load your content and execute it within its own thread [74].

Electron is used by companies, large and small, to build desktop applications. It was originally developed as the foundation for GitHub's Atom text editor. Facebook released Nuclide as a package on top of Atom that turns the text editor into a full-fledged integrated development environment (IDE) with first-class support for working with React Native, Hack, and Flow projects. Microsoft also uses Electron for its cross-platform Visual Studio Code editor, which runs on macOS, Windows, and Linux [75]. Electron enables developers to use web technologies such as HTML5, CSS and JavaScript. These technologies allows a single application to work on multiple operating systems using the same markup language of websites and requires a minimal level of investment in technical knowledge and time [76].

In this work, we apply hybrid approach to build a desktop application as an alternative for the current DSSAT Shell. There are many advantages to follow this approach such as reusability, portability and flexibility. The details on how we structure this work are seen in the sections 5.4 and 5.5.

5.4 SOFTWARE ARCHITECTURE

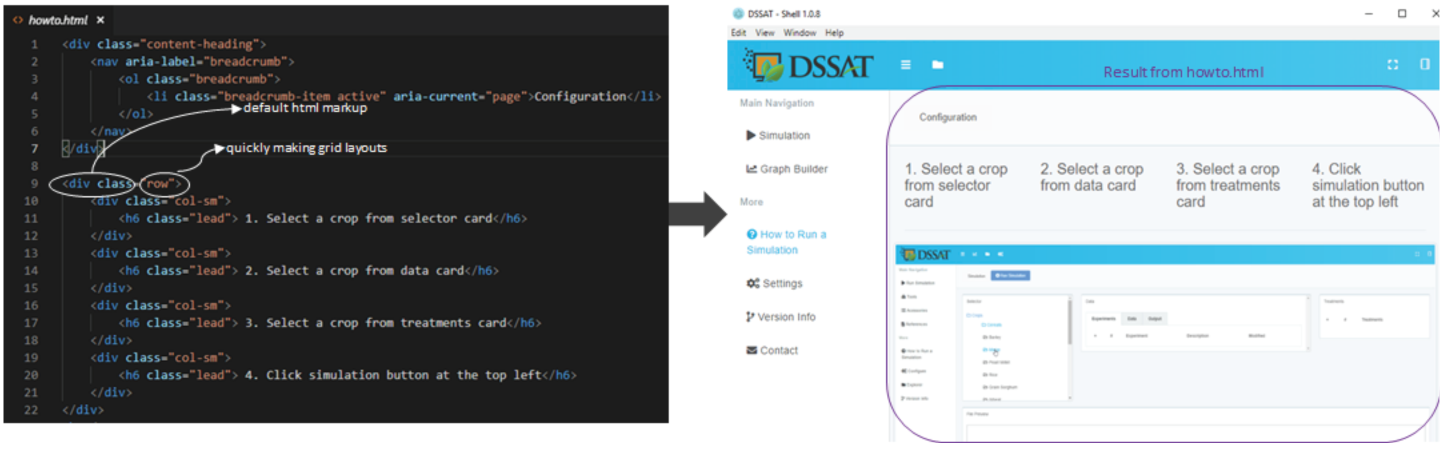
The DSSAT-Shell architecture is divided into three parts, GUI implementation, GUI rendering and GUI integration with DSSAT-CSM. In this chapter, we describe how the DSSAT-Shell user interface is implemented as well as how the user interface is rendered on the user's computer and finally how the integration with DSSAT-CSM is done.

5.4.1 GUI implementation

Application developers in general, and website providers in particular, currently have to deal with the increased range of new devices and diversity of interface characteristics [77]. A new trend is therefore responsive *web design* which means to build the layout of the interface on fluid grids that can dynamically adapt to diverse viewing environments [78].

Responsive interfaces are challenge for any application development team. The Bootstrap framework, which has become one of the standout framework was developed at Twitter in 2010 [79], helping developer build responsive interfaces by combining a set of libraries such as CSS, fonts and JavaScript. Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases [80]. The DSSAT-

Shell combines HTML and Bootstrap to build the visual interface, as seen in Figure 22. It also uses JavaScript, to handle user interface events as well as create the DSSAT-CSM integration layer.



```html
<div class="content-heading">
    <nav aria-label="breadcrumb">
        <ol class="breadcrumb">
            <li class="breadcrumb-item active" aria-current="page">Configuration</li>
        </ol>
    </nav>
</div>

<div class="row">
    <div class="col-sm">
        <h6 class="lead"> 1. Select a crop from selector card</h6>
    </div>
    <div class="col-sm">
        <h6 class="lead"> 2. Select a crop from data card</h6>
    </div>
    <div class="col-sm">
        <h6 class="lead"> 3. Select a crop from treatments card</h6>
    </div>
    <div class="col-sm">
        <h6 class="lead"> 4. Click simulation button at the top left</h6>
    </div>
</div>
```

Figure 22. Snippet from the DSSAT-Shell "how to" page. In the left side, it combines HTML markups and bootrap classes. In the right side is the result of the code in the right side.

5.4.2 GUI rendering

There are many approaches to manage application lifecycle and communication between processes. The comunication betwen main processes and renderes is done in a separate I/O thread. Messages to and from the views have to be proxied over to the main thread using a ChannelProxy. The advantage of this scheme is that resource requests (for web pages, etc.), which are the most common and performance critical messages, can be handled entirely on the I/O thread and not block the user interface [81]. There are three main steps that happens once the application is started as shown in Figure 23.

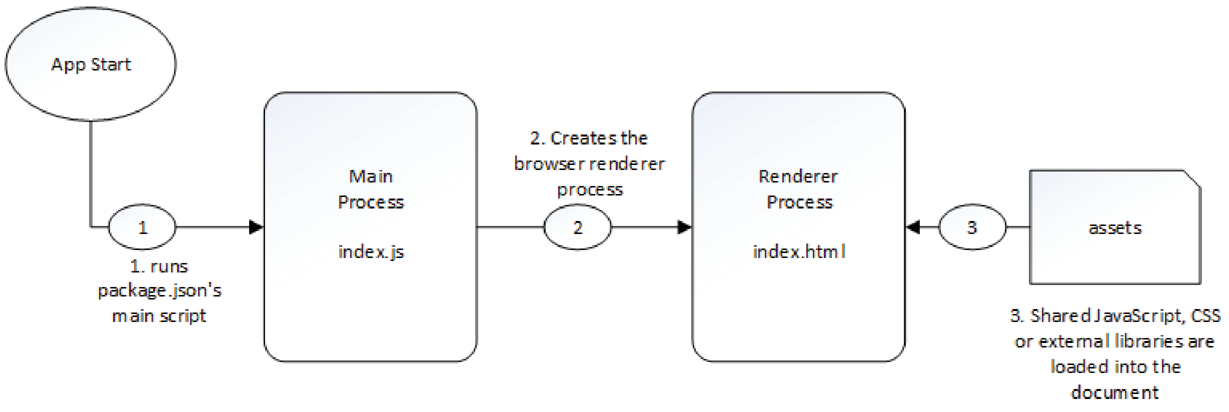
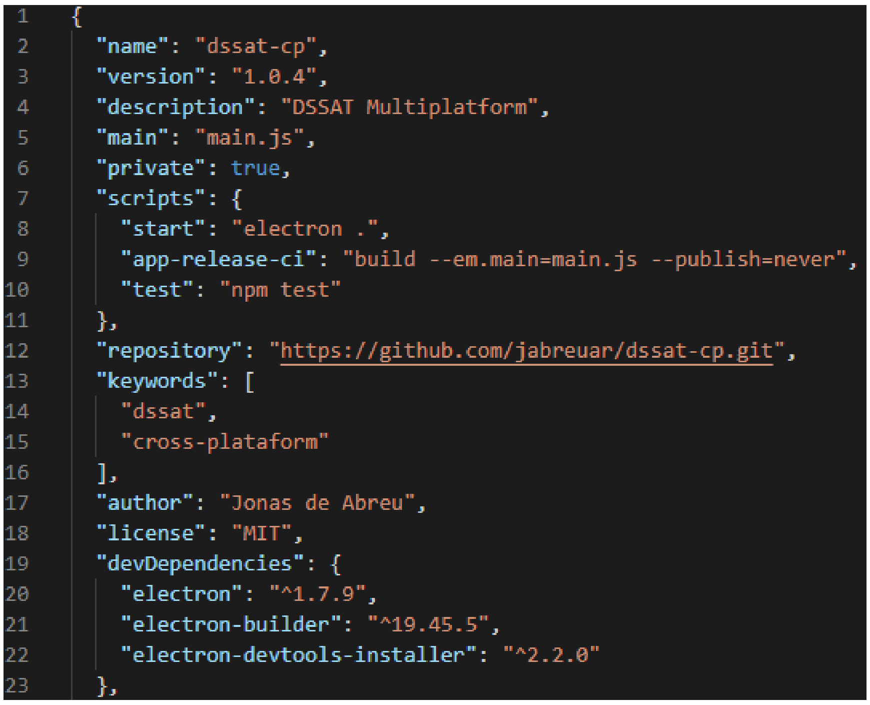

App Start
1
1. runs package.json's main script
Main Process
index.js
2
2. Creates the browser renderer process
Renderer Process
index.html
3
assets
3. Shared JavaScript, CSS or external libraries are loaded into the document

Figure 23. Steps to display GUI by rendering web pages through IPC.

Electron needs an entry point which is a JavaScript file that contains some definitions about the application. Typically, this entry point is configured in the package.json, seen in Figure 24, that points to an entry point file. This entry point file is used to load any modules required by the package [82].



```
1  {
2    "name": "dssat-cp",
3    "version": "1.0.4",
4    "description": "DSSAT Multiplatform",
5    "main": "main.js",
6    "private": true,
7    "scripts": {
8      "start": "electron .",
9      "app-release-ci": "build --em.main=main.js --publish=never",
10     "test": "npm test"
11   },
12   "repository": "https://github.com/jabreuar/dssat-cp.git",
13   "keywords": [
14     "dssat",
15     "cross-plataform"
16   ],
17   "author": "Jonas de Abreu",
18   "license": "MIT",
19   "devDependencies": {
20     "electron": "^1.7.9",
21     "electron-builder": "^19.45.5",
22     "electron-devtools-installer": "^2.2.0"
23   },
```

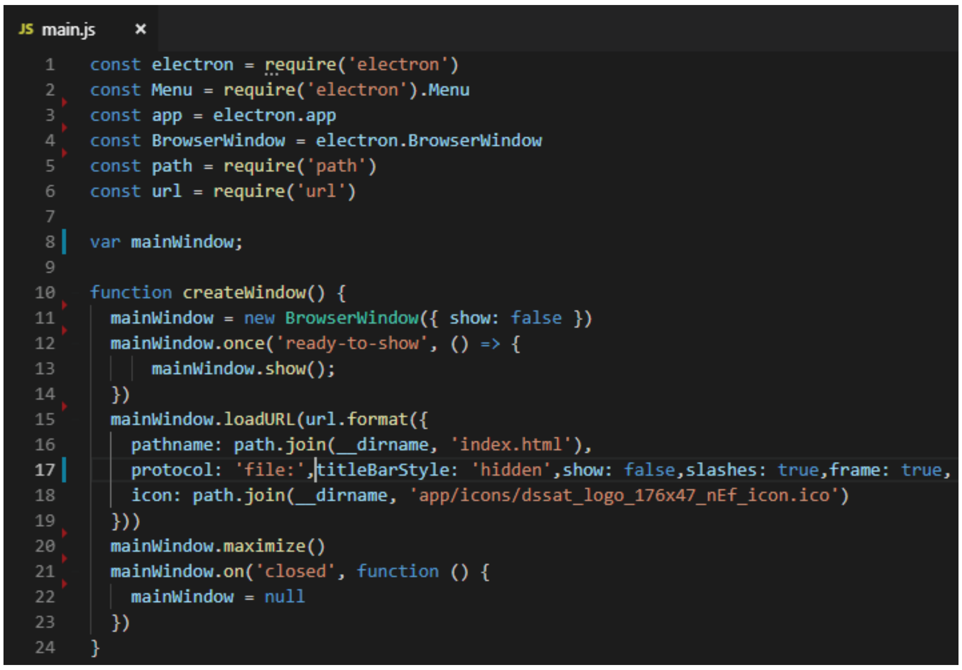
Figure 24. Snippet from the DSSAT-Shell package.json. The line 5 correspond to the entry point file that is accessible used when the application loaded.

The *main.js* file is the process entry point. This file sets how the application will handle the main window. The menu of the application is also created on the main file. Finally, we define what is the HTML file to load in the main window, as seen in Figure 25.

5.4.3 DSSAT-CSM integration

The Decision Support System for Agrotechnology Transfer (DSSAT) is a computer software that combines many crop models [65]. It is on-premises software, installed and runs on computers on the premises of the user or organization using the software. A series of files are copied on the user's machine once DSSAT installed, including the DSSAT-CSM executable which is used to run models through command line. The crop models are executed through command line interpreter. If the model is run using the existing DSSAT shell, these command lines are transparent to the user. Different types of applications are accomplished in DSSAT-CSM. The mode is specified as a command line argument when the model is run. We use the basic mode provides for interactive sensitivity analysis and comparison of simulated vs. observed field data [83].

The new DSSAT-Shel solution, as seen in Figure 26, has three main components, the renderes, the native UI and the frameworks to access the file system. Electron renders the user interface and manage the comunicattion betwen the browser window instances. The native UI implements the interface components and handle user actions such as run simulation and



```
main.js ×
 1  const electron = require('electron')
 2  const Menu = require('electron').Menu
 3  const app = electron.app
 4  const BrowserWindow = electron.BrowserWindow
 5  const path = require('path')
 6  const url = require('url')
 7
 8  var mainWindow;
 9
10  function createWindow() {
11    mainWindow = new BrowserWindow({ show: false })
12    mainWindow.once('ready-to-show', () => {
13        mainWindow.show();
14    })
15    mainWindow.loadURL(url.format({
16      pathname: path.join(__dirname, 'index.html'),
17      protocol: 'file:',titleBarStyle: 'hidden',show: false,slashes: true,frame: true,
18      icon: path.join(__dirname, 'app/icons/dssat_logo_176x47_nEf_icon.ico')
19    }))
20    mainWindow.maximize()
21    mainWindow.on('closed', function () {
22      mainWindow = null
23    })
24  }
```

Figure 25. Snippet from the main.js of the DSSAT-Shell application. The line 16 correspond to the entry point HTML that is loaded in the renderer processes. There are others configuration that is defined in the main.js file such as application icon (line 18) and how we handle the main window (line 20).

display experiments for a given crop and others. Once the inputs are retrieved, the UI send it to jDSSAT, which is a library for process DSSAT-CSM files. The jDSSAT uses Node.js modules, which are any stand-alone JavaScript that run in the node enviroment to perform general or specific actions [84]. The jDSSAT uses file system module from to access the to access DSSAT work directoty on the user machine.

With this design, the core DSSAT-CSM is completely separated from the user interface implementation, which in turn, is separeted fromthe renderization to the user viewing. This aproach aimed avoid having to touch in many files when a change is required, also it helps in the separation of concerns because the modules of the solution are isolated.

5.5 SOFTWARE FUNCTIONALITIES AND ILLUSTRATIVE EXAMPLES

This section discusses the functionalities of the current release of DSSAT-Shell and also shows examples of the new user interface.

5.5.1 Functionalities

The current DSSAT-Shell has many external tools and accessories, as we can see in Figure 20. These tools are accessible through the shell and supported in Windows only, also most of them were built using Visual Basic which is a legacy product from Microsoft. Therefore,

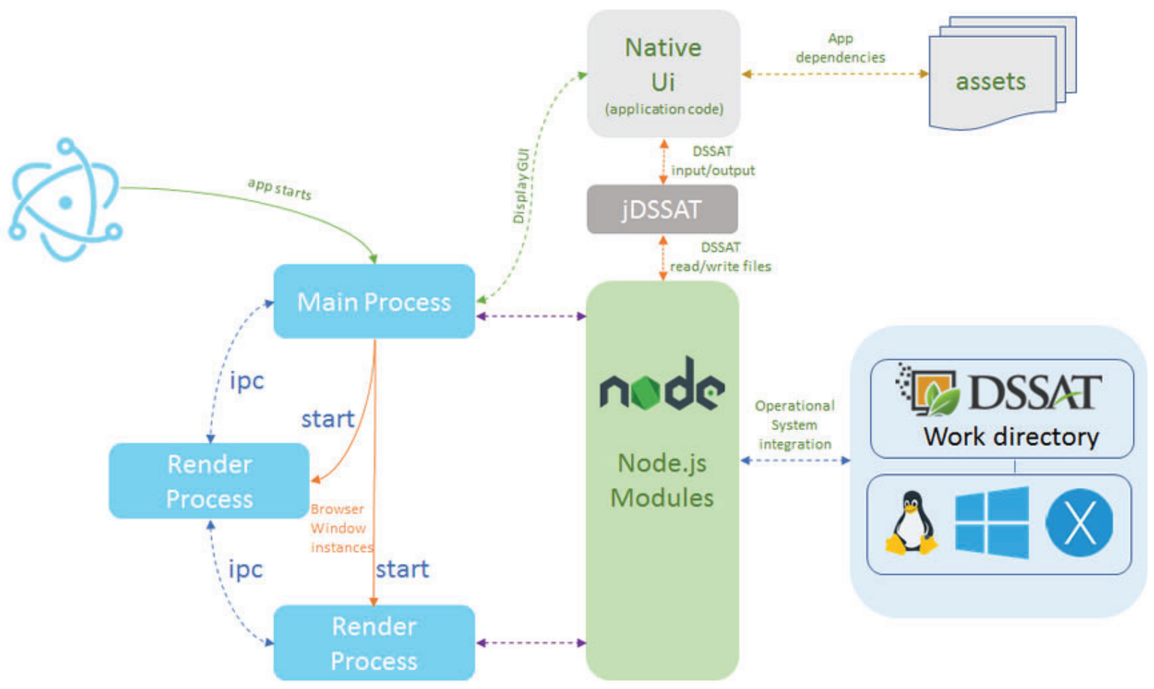


Figure 26. Overview of the components and modular structure of the DSSAT-Shell implementation. The electron main process start the renderers. Each render is a page that loads according UI events. The user interface uses jDSSAT module to integrate with DSSAT thought node js modules.

the new DSSAT should support external tools, so the same tools menu will be available for the user in the new shell.

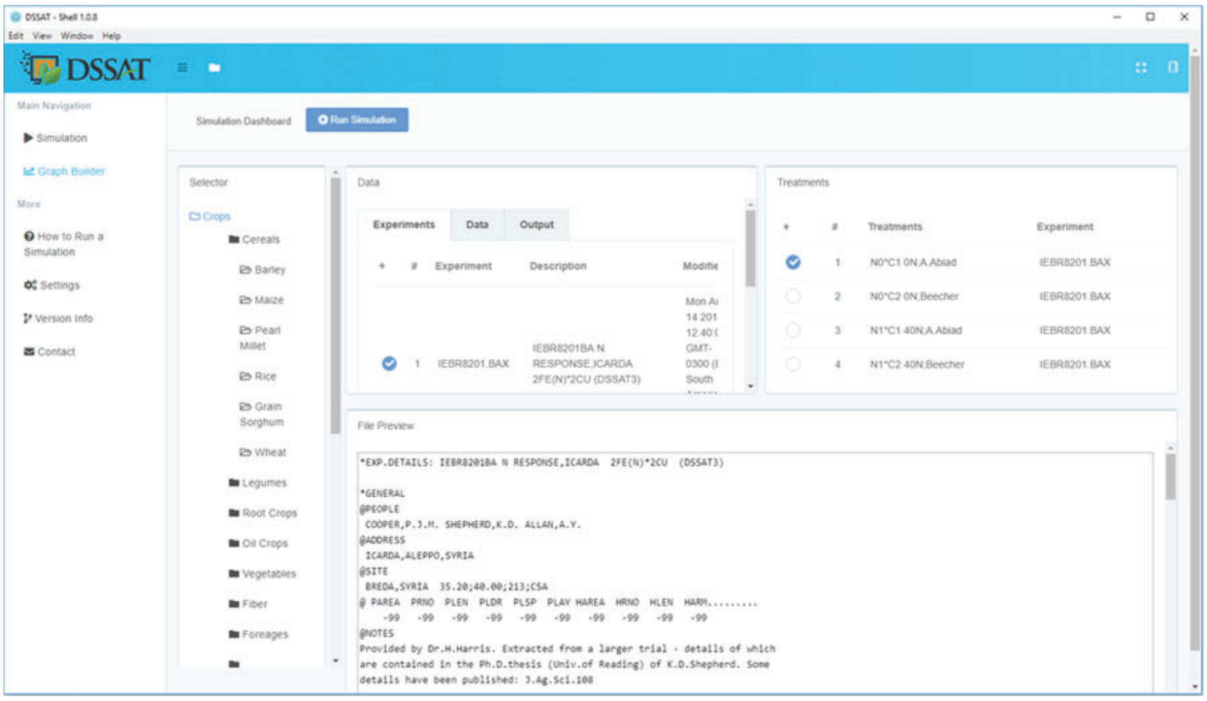
The proposed user interface should implement basic functionalities such as running a simulation and provide a visualization for the simulation results. The simulation should include options to the user to select required inputs for a DSSAT simulation such as crop, experiments and treatments. There is also a need of display the experiments and treatment file preview. For displaying results, the new shell should read DSSAT output files content to display options to select variables and experiements to plot. In case there are available data as part of the simulation, it should be displayed in the graph result as well.

There are some enhancements over the existing shell, as follows:

- The installers for future versions are easily generated.
- Flexibility to add new crop models.
- Usability of the same code base in different environments.

5.5.2 Examples

In this section we provide some examples of the new DSSAT-Shell. As we can see in the Figure 27, the main page is divided into 4 cards. The selector card displays all crops available. The data card shows all experiments for the crop that the user selected. The tool allows the user to select multiple experiments for a simulation. The treatments card, at the right in the user interface displays all treatments available for the experiments selected. Finally, at the botton of the tool there will be a file preview, that will display the last experiment file selected by the user. There is also a Run Simulation button at the top of the interface, after the user choose all arguments for a simulation, then this button is used to run the commands in DSSAT.



+	#	Experiment	Description	Modifie
✓	1	IEBR8201.BAX	IEBR8201BA N RESPONSE,ICARDA 2FE(N)*2CU (DSSAT3)	Mon A 14 201 12:40:(GMT-0300 (I South

+	#	Treatments	Experiment
✓	1	N0*C1 0N;A.Abiad	IEBR8201.BAX
	2	N0*C2 0N;Beecher	IEBR8201.BAX
	3	N1*C1 40N;A.Abiad	IEBR8201.BAX
	4	N1*C2 40N;Beecher	IEBR8201.BAX

```
*EXP.DETAILS: IEBR8201BA N RESPONSE,ICARDA  2FE(N)*2CU  (DSSAT3)

*GENERAL
@PEOPLE
 COOPER,P.J.M. SHEPHERD,K.D. ALLAN,A.Y.
@ADDRESS
 ICARDA,ALEPPO,SYRIA
@SITE
 BREDA,SYRIA  35.20;40.00;213;CSA
@ PAREA  PRNO  PLEN  PLDR  PLSP  PLAY HAREA  HRNO  HLEN  HARM.........
    -99   -99   -99   -99   -99   -99   -99   -99   -99   -99
@NOTES
Provided by Dr.H.Harris. Extracted from a larger trial - details of which
are contained in the Ph.D.thesis (Univ.of Reading) of K.D.Shepherd. Some
details have been published: J.Ag.Sci.108
```

Figure 27. Screenshot from the main page of the DSSAT-Shell. There are four main components that are the selector, data, treatments and preview. The selector component display the crops available in DSSAT, once the user select the crop, the experiments are loaded under data components. After the user selectes the experiments the treatments are loaded. The preview component display the FILEX (experiment file) content.

Once the simulation is done, the user can visualize the simulation results. As shown in Figure 28, there is a wizard in the graph builder to help users to plot a graph. In the left, it shows the crop selection; if the user ran a simulation, this field option is automatically selected. In the right, we see the option to select from each output file the user wants to select the variable values.

Finally, the last step is to analyze the results in a graph, as we can see in Figure 29. The graph shows the data based on what the user selected on the previous steps.

5.6 FINAL CONSIDERATIONS

In this chapter, we have presented an user interface proposal for the current DSSAT-Shell, flexible, cross-plataform, modular and powerful that provides a modern development ap-

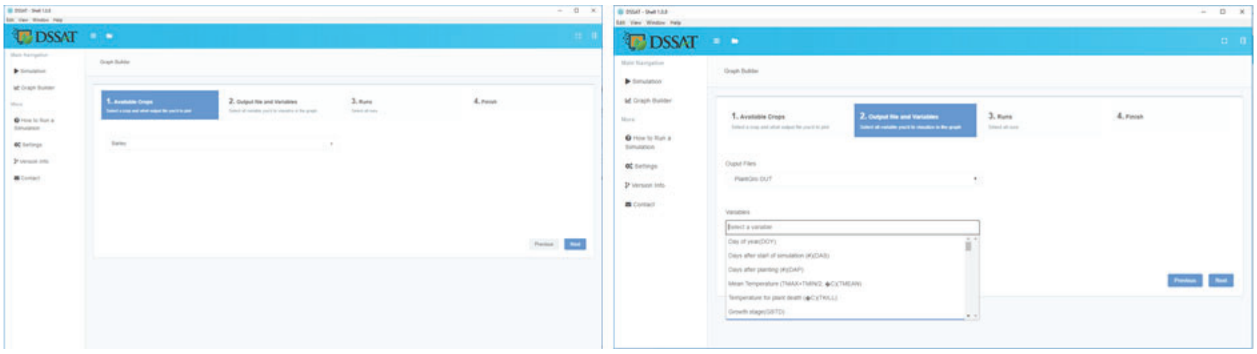


Figure 28. Screenshot from DSSAT-Shell graph builder. It displays options for the user to select what file he wants and its variables.

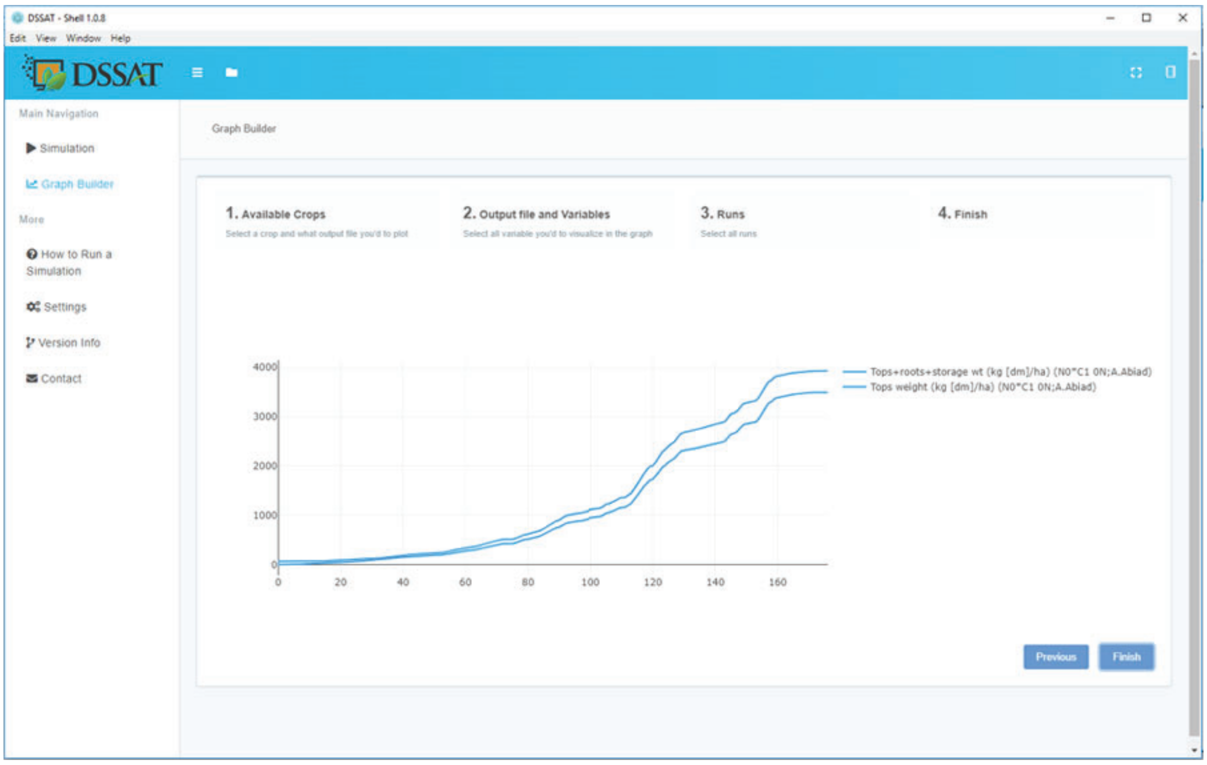


Figure 29. Graph plot based on the variables that the selected in the Output and Variables step.

proach. The implementation of the Shell also highlights some benefits in terms of longevity, since the current technologies being used by the current DSSAT-Shell will no longer be supported.

The Shell is an ongoing project. We intend to apply the same development approach in other DSSAT tools used by the community such as Crop Rotation, XBuilder and Sensitive Analysis. We also plan to extend its distribution by implementing update management for future releases of the DSSAT-CSM.

Currently the DSSAT software is distributed individually, users have a copy of the DSSAT software and a token to install it. The DSSAT-Lite approach opens up the possibility to improve user management access, a new page can be created in DSSAT-Lite for the user to registering in the tool. Ultimately, as we use web-technologies, we intend to collect metrics to understand user needs to improve even more the experience in the tool

6. CONCLUSIONS

6.1 FINAL CONSIDERATIONS

The jDSSAT module was very useful in the development of the tools presented in this work. In addition, to abstract the complexity of processing DSSAT files, it integrates seamlessly with other environments, for example, R. The cloud development approach for DSSAT-Lite shown very good results, the concurrency issues that might be a problem for simulations has not been reproduced yet. Finally, the DSSAT-Shell was tested in Windows, Linux and MacOS and presented the same user experience in all enviroments.

We presented the results of this work in the 10th DSSAT Development Sprint at the University of Florida. The DSSAT community were receptive. Many participants gave feedback to improve further the tools. Also, some oportunities were found such as generating reports based on the user's activities to collect information such as what are the most crop used and what the most used variables to plot graphs.

In this context, it is concluded that the objective of this work was achieved, since the tools presented have modern development approach that can be extended and used for new implementations. Also, the tools developed as part of this work presented the same behaviour as the original DSSAT-Shell.

6.2 FUTURE WORK

As part of this work were developed: a module to read, write and process DSSAT-CSM files, and they were used in the DSSAT-Shell and DSSAT-Lite tools as the midleware for DSSAT integration. The following activities were discussed aiming at the evolution of the tools.

Initially, we understand that DSSAT comunity have good knowledge of the R language and its enviroment. Thus, there is an oportunity to develop a jDSSAT R package that will provide the same functions available in JavaScript, this way R users will be able to use R tool to visualize simulation results without worrying about processing DSSAT files.

Another important contribution would be to implement a more complete installer for the DSSAT-Shell, since it requires previously DSSAT installation at the user machine. This way, would be possible to create automatically updates when a new DSSAT release is published.

REFERENCES

[1] BANNAYAN, M.; CROUT, N.; HOOGENBOOM, G. Application of the ceres-wheat model for within-season prediction of winter wheat yield in the united kingdom. *Agronomy Journal - AGRON J*, v. 95, 01 2003.

[2] GRAVES, A. R. et al. Crop simulation models as tools in computer laboratory and classroom-based education. *Journal of Natural Resources and Life Sciences Education*, Citeseer, v. 31, p. 48–58, 2002.

[3] LAZZARETTI, A. T. et al. Integração de banco de dados e modelos de simulação de culturas para estimular o impacto de mudanças do clima no rendimento de grãos e na severidade da giberela em trigo. Universidade de Passo Fundo, 2013.

[4] PAVAN, W. et al. Técnicas de engenharia de software aplicadas à modelagem e simulação de doenças de plantas. Universidade de Passo Fundo, 2007.

[5] VERHAGEN, A.; CONIJN, J.; SCHAPENDONK, A. *Quickscan of simulation models*. [S.l.], 2001.

[6] SARKAR, R.; KAR, S. Evaluation of management strategies for sustainable rice–wheat cropping system, using dssat seasonal analysis. *The Journal of Agricultural Science*, Cambridge University Press, v. 144, n. 5, p. 421–434, 2006.

[7] FOUNDATION, D. *DSSAT Tools*. 2018. https://dssat.net/tools. [Accessed 24 March 2019].

[8] JONES, J. et al. The DSSAT cropping system model. *European Journal of Agronomy*, v. 18, n. 3, p. 235–265, 2003. ISSN 1161-0301. https://doi.org/10.1016/S1161-0301(02)00107-7.

[9] PORTER, C. H.; WILKENS, P. W. *DSSAT v4.5 Overview*. [S.l.: s.n.]. v. 1. ISBN 1886684065.

[10] SAMETINGER, J. *Software engineering with reusable components*. [S.l.]: Springer Science & Business Media, 1997.

[11] SHANNON, R. E. Introduction to the art and science of simulation. In: IEEE COMPUTER SOCIETY PRESS. *Proceedings of the 30th conference on Winter simulation*. [S.l.], 1998. p. 7–14.

[12] CHWIF, L.; MEDINA, A. C. *Modelagem e simulação de eventos discretos*. [S.l.]: Afonso C. Medina, 2006.

[13] FORRESTER, J. W. *Principles of systems: text and workbook*. [S.l.]: Wright-Allen Press, 1969.

[14] HOOGENBOOM, G. Contribution of agrometeorology to the simulation of crop production and its applications. *Agricultural and forest meteorology*, Elsevier, v. 103, n. 1, p. 137–157, 2000.

[15] SARKAR, R. et al. Use of DSSAT to model cropping systems. *CAB Reviews: perspectives in agriculture, veterinary science, nutrition and natural resources*, CABI, v. 4, n. 025, p. 1–12, 2009. https://doi.org/10.1079/PAVSNNR20094025.

[16] DSSAT Foundation 2016. 2016. Disponível em: <http://http://dssat.net, urlaccessdate = Jun. 16, 2017>.

[17] JONES, J. W. et al. Decision support system for agrotechnology transfer: DSSAT v3. In: ____. *Understanding Options for Agricultural Production*. Dordrecht: Springer Netherlands, 1998. p. 157–177. https://doi.org/10.1007/978-94-017-3624-4_8.

[18] HUNT, L. et al. Gencalc: software to facilitate the use of crop models for analyzing field experiments. *Agronomy Journal*, American Society of Agronomy, v. 85, n. 5, p. 1090–1094, 1993.

[19] FARIA, R. T. d.; BOWEN, W. T. Evaluation of dssat soil-water balance module under cropped and bare soil conditions. *Brazilian Archives of Biology and Technology*, SciELO Brasil, v. 46, n. 4, p. 489–498, 2003.

[20] W3C. *W3C*. 2017. Disponível em: <https://www.w3.org/>.

[21] MOZILLA. *Mozilla*. 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>.

[22] Google Inc. *V8 Google's open source high-performance JavaScript and WebAssembly engine*. [S.l.], 2018. https://v8.dev/. [Accessed 10 June 2018].

[23] WEBKIT. *WebKit*. 2017. Disponível em: <https://webkit.org/project/>.

[24] OPERA. *carakan*. 2009. Disponível em: <https://dev.opera.com/blog/carakan/>.

[25] MICROSOFT. *chakra*. 2017. Disponível em: <https://blogs.windows.com/msedgedev/2015/12/05/open-source-chakra-core/>.

[26] FREEMAN, A. *Pro Angular*. [S.l.]: Springer.

[27] TILKOV, S.; VINOSKI, S. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, IEEE, v. 14, n. 6, p. 80–83, 2010.

[28] INITIATIVE, O. S. *BSD License*. 2017. Disponível em: <https://opensource.org/licenses/bsd-license.php>.

[29] GOOGLE. *The Chromium Projects*. https://www.chromium.org/. [Accessed 04 January 2019].

[30] PYDSSAT. *About pyDSSAT*. 2015. Disponível em: <http://xiaoganghe.github.io/pyDSSAT/doc/about_pyDSSAT.html>. Acesso em: Jun. 14, 2017.

[31] HE LIQING PENG, H. S. X. *pyDSSAT Documentation*. pyDSSAT Documentation, 2015. Disponível em: <http://xiaoganghe.github.io/pyDSSAT/doc/_downloads/pyDSSATdoc.pdf>.

[32] PYDSSAT. *pyDSSAT API*. 2017. Disponível em: <http://xiaoganghe.github.io/pyDSSAT/doc/api.html>. Acesso em: Jun. 14, 2017.

[33] BOOTE, K. J. et al. The role of crop systems simulation in agriculture and environment. *International Journal of Agricultural and Environmental Information Systems (IJAEIS)*, IGI Global, v. 1, n. 1, p. 41–54, 2010. https://doi.org/10.4018/jaeis.2010101303.

[34] LIU, H. et al. Using the DSSAT model to simulate wheat yield and soil organic carbon under a wheat-maize cropping system in the north china plain. *Journal of Integrative Agriculture*, v. 16, n. 10, p. 2300–2307, 2017. ISSN 2095-3119. https://doi.org/10.1016/S2095-3119(17)61678-2.

[35] DOURADO-NETO, D. et al. Principles of crop modeling and simulation: I. uses of mathematical models in agricultural science. *Scientia Agricola*, SciELO Brasil, v. 55, n. SPE, p. 46–50, 1998.

[36] ROSE, D. C. et al. Decision support tools for agriculture: Towards effective design and delivery. *Agricultural systems*, Elsevier, v. 149, p. 165–174, 2016.

[37] SARKAR, R. Decision support systems for agrotechnology transfer. In: ____. *Organic Fertilisation, Soil Quality and Human Health*. Dordrecht: Springer Netherlands, 2012. p. 263–299. ISBN 978-94-007-4113-3. https://doi.org/10.1007/978-94-007-4113-3_10.

[38] WILKENS, P. W. et al. *DSSAT v4 Data Management and Analysis Tools*. [S.l.]: International Consortium for Agricultural Systems Applications, 2004. v. 2. 177 p. ISBN 1-886684-07-3.

[39] HE, X.; PENG, L.; SUN, H. *pyDSSAT Documentation Release 1.0*. [S.l.], 2015. http://xiaoganghe.github.io/pyDSSAT/doc/index.html. [Accessed 16 September 2018].

[40] LOZZA, H. *Dasst: Tools for Reading, Processing and Writing DSSAT files, R package version 0.3.3*. Buenos Aires, Argentina, 2017. https://github.com/hlozza/Dasst. [Accessed 10 July 2018].

[41] FOWLER, M. Separating user interface code. *IEEE software*, IEEE, v. 18, n. 2, p. 96–97, 2001.

[42] GAMMA, E. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education India, 1995.

[43] Node.js Foundation. *Node.js*. [S.l.], 2018. https://nodejs.org/en/. [Accessed 24 September 2018].

[44] HERRON, D. *Node Web Development*. [S.l.]: Packt Publishing Ltd, 2013.

[45] YANG, J.; HUFFMAN, E. T. EasyGrapher: software for graphical and statistical validation of DSSAT outputs. *Computers and Electronics in Agriculture*, v. 45, n. 1, p. 125–132, 2004. ISSN 0168-1699. https://doi.org/10.1016/j.compag.2004.06.006.

[46] WILSON, J. *Node.js 8 the Right Way: Practical, Server-side Javascript that Scales*. [S.l.]: Pragmatic Bookshelf, 2018. ISBN 978-1-68050-195-7.

[47] npm Inc. *Node package manager*. [S.l.], 2018. https://www.npmjs.com/. [Accessed 10 July 2018].

[48] SCHROEDER, R.; BRÜGGER, N.; COWLS, J. Historical web as a tool for analyzing social change. *Second International Handbook of Internet Research*, Springer, p. 1–16, 2018.

[49] SHAHZAD, F. Modern and responsive mobile-enabled web applications. *Procedia Computer Science*, v. 110, p. 410 – 415, 2017. ISSN 1877-0509. 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops. Disponível em: <http://www.sciencedirect.com/science/article/pii/S187705091731284X>.

[50] O'REILLY, T. *What is web 2.0*. 2005.

[51] SARKAR, R. Use of DSSAT to model cropping systems. *CAB Reviews: Perspectives in Agriculture, Veterinary Science, Nutrition and Natural Resources*, v. 4, n. 025, p. 1–12, 2009. ISSN 17498848.

[52] HOOGENBOOM, G. et al. Decision support system for agrotechnology transfer version 4.0. *University of Hawaii, Honolulu, HI (CD-ROM)*, 2004.

[53] FOUNDATION GAINESVILLE, F. U. D. *Decision Support System for Agrotechnology Transfer (DSSAT) Version 4.7*. Disponível em: <https://dssat.net/about>.

[54] WHITE, J. W. et al. Integrated description of agricultural field experiments and production: The ICASA version 2.0 data standards. *Computers and Electronics in Agriculture*, v. 96, p. 1–12, 2013. ISSN 0168-1699. https://doi.org/10.1016/j.compag.2013.04.003.

[55] FOUNDATION, N. *Express*. Disponível em: <https://expressjs.com/en/guide/writing-middleware.html>.

[56] MICROSOFT. *TypeScript*. Disponível em: <https://www.typescriptlang.org/>.

[57] GOOGLE. *Angular*. Disponível em: <https://angular.io/>.

[58] ARORA, C.; HENNESSY, K. *Angular 6 by Example: Get up and running with Angular by building modern real-world web apps*. [S.l.]: Packt Publishing Ltd, 2018.

[59] LETT, J. Bootstrap 4 quick start: A beginners guide to building responsive layouts with bootstrap 4. Bootstrap Creative, 2018.

[60] SOFTWARE, P. *Jasmine*. Disponível em: <https://jasmine.github.io/>.

[61] FIELDING, R. T.; TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. v. 7.

[62] W3SCHOOLS. *HTML5 Web Storage*. Disponível em: <https://www.w3schools.com/html/html5_webstorage.asp>.

[63] CROCKFORD, D. *The application/json media type for javascript object notation (json)*. [S.l.], 2006.

[64] SIEVERT, C. *plotly for R*. [S.l.], 2018. https://plotly-book.cpsievert.me. [Accessed 15 June 2018].

[65] YANG, J. et al. Easygrapher: software for data visualization and statistical evaluation of dssat cropping system model and the canb model. *International Journal of Computer Theory and Engineering*, IACSIT Press, v. 6, n. 3, p. 210, 2014.

[66] FOUNDATION, D. *DSSAT About*. https://dssat.net/about. [Accessed 13 February 2019].

[67] BALENA, F.; FAWCETTE, J. *Programming Microsoft Visual Basic 6.0*. [S.l.]: Microsoft press Washington, 1999. v. 1.

[68] MICROSOFT. *Support Statement for Visual Basic 6.0 on Windows*. https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6-support-policy. [Accessed 13 February 2019].

[69] FERREIRA, C. M. S. et al. An evaluation of cross-platform frameworks for multimedia mobile applications development. *IEEE Latin America Transactions*, v. 16, n. 4, p. 1206–1212, April 2018. ISSN 1548-0992.

[70] HEITKÖTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Evaluating cross-platform development approaches for mobile applications. In: CORDEIRO, J.; KREMPELS, K.-H. (Ed.). *Web Information Systems and Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 120–138. https://doi.org/10.1007/978-3-642-36608-6_8. [Accessed 16 May 2018].

[71] THAKARE, B. S. et al. State of art approaches to build cross platform mobile application. *International Journal of Computer Applications*, v. 107, n. 20, p. 22–23, 2014.

[72] RAJ, C. R.; TOLETY, S. B. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: IEEE. *India Conference (INDICON), 2012 Annual IEEE*. [S.l.], 2012. p. 625–629.

[73] COMMUNITY, E. *Build cross platform desktop apps with JavaScript, HTML, and CSS*. https://electronjs.org/. [Accessed 01 February 2019].

[74] GRIFFITH, C.; WELLS, L. *Electron: From Beginner to Pro: Learn to Build Cross Platform Desktop Applications Using Github's Electron*. [S.l.]: Apress, 2017.

[75] KINNEY, S. *Electron in Action*. [S.l.]: Manning, 2018. 6,7 p.

[76] ANDRADE, P. R. D. et al. Cross platform app: a comparative study. *arXiv preprint arXiv:1503.03511*, 2015.

[77] NEBELING, M.; NORRIE, M. C. Responsive design and development: methods, technologies and current issues. In: SPRINGER. *International Conference on Web Engineering*. [S.l.], 2013. p. 510–513.

[78] MARCOTTE, E. *Responsive web design: A book apart n° 4*. [S.l.]: Editions Eyrolles, 2017.

[79] CLETUS, W. D.; KAKANDAR, A.; PAUL, C. N. Responsive web design frameworks: A review. 2017.

[80] OTTO, J. T. M. *Bootstrap*. https://getbootstrap.com/docs/3.3/css/#grid. [Accessed 13 February 2019].

[81] GOOGLE. *The Chromium IPC*. https://www.chromium.org/developers/design-documents/inter-process-communication. [Accessed 11 February 2019].

[82] KULA, R. G. et al. On the impact of micro-packages: An empirical study of the npm javascript ecosystem. *arXiv preprint arXiv:1709.04638*, 2017.

[83] JONES, J. et al. Dssat cropping system model. *European Journal of Agronomy*, v. 18, p. 235–265, 01 2003.

[84] CANTELON, M. et al. *Node. js in Action*. [S.l.]: Manning Greenwich, 2014.