

Design Document for Kairos Constraint-Based Scheduling Software System

Tyler Chapman, Nate Crandall, Vinh Dang, Vince Oveson, Tony Tuttle

December 18, 2014

Contents

1	Executive Summary	2
1.1	Overview	2
1.2	Features and Components	2
1.3	Justification	2
2	Background	2
2.1	Overview	2
2.1.1	Similar Ideas	3
2.1.2	How Kairos is Different	3
2.2	Required Technology	3
2.2.1	Core Web Service	3
2.2.2	SoC Module	3
2.2.3	API	4
2.3	Assets and Engines	4
2.4	Software/Hardware Requirements for Users	4
3	Requirements Analysis	4
3.1	System Architecture	4
3.1.1	Constraint Solver	4
3.1.2	App Server	4
3.1.3	Apache Server	4
3.2	Personnel	5
3.3	System Features	5
3.3.1	Basic Features	6
3.3.2	Planned Features	6
3.3.3	Advanced Features	7
4	Tools and Techniques	7
5	Time Line	9
Appendix A	Use Cases	10
A.1	Entering Data Manually to Create a New Schedule	10
A.2	Uploading a CSV File to Create a New Schedule	10
A.3	Exporting Schedule as a CSV File	11
A.4	Modifying a Proposed Schedule	11
A.5	Comparing Two Schedules	11
A.6	Administrative Tasks	11
A.7	API use	12
Appendix B	UI Sketches	13

List of Figures

1	Overview of the System	5
2	Kairos landing screen	13
3	User schedule selection dashboard	14
4	User data entry screen for classes	15
5	User data entry screen for rooms	16
6	Visualization of a schedule	17
7	Single day visualization of a schedule	17

1 Executive Summary

1.1 Overview

Scheduling problems are ubiquitous. Individuals, teams, organizations, departments and companies all must solve scheduling problems of various levels of complexity. Our tool aims to address the needs of some of these users, aiming primarily at small to medium sized entities, and those with sufficient technical expertise to work with an API.

The goal of the Kairos scheduling software is to provide a customizable, open-source, web-based scheduling tool to solve a wide range of scheduling problems. Our software system will be accessible through a public website where users may build, modify, and maintain their solutions. The system will be capable of solving various types of scheduling problems for various user needs.

1.2 Features and Components

At the core of Kairos will be a constraint solver that is set up to specifically solve scheduling problems. The user will supply details of their scheduling problem such as the events to schedule, the resources available, and any constraints. The solver will determine first if a schedule exists for all of the events, given the resources and constraints. If so, it will assign a time and place for each event and return this schedule to the user. If a schedule is not possible, it will provide as informative a message as possible to the user indicating as much.

On the other end of the tool will be the user interface. Scheduling is a complex problem that produces data that can have several dimensions. Representing this data in a way that does not alienate users is a significant challenge. We will focus on making our visualizations of the complex data as intuitive, aesthetically appealing, and simple as possible.

For those users who prefer flexibility and want to design their own interfaces, we will offer an exposed API so that they are able to access the solver at the core while building their own UIs and visualization tools around the data. Each user's needs will be unique. Offering them this option will improve the overall usability of the tool.

The success of the customizability will depend a clear and thorough API. A great piece of software may lose users if it is not clear how to leverage the software. We intend to encourage a large user base by putting a lot of emphasis on creating a strong API.

By hosting the solver as well as user data on our server, we will be able to make Kairos a web-based tool. This is another means by which we can attract a wider user base. Making our tool web-based will eliminate the need for users to download and run an application, will allow users to share data with others seamlessly, and will prevent users from using out-of-date software.

1.3 Justification

When Kairos is complete, it can offer a free, customizable tool that could save organizations of various sizes a great deal of time and money. If we are able to attract a sufficient base of users, we will have a wealth of feedback to make the tool even better.

2 Background

2.1 Overview

The task of creating optimal schedules is a problem that event planners, groups, teams, and other organizations encounter on a regular basis. The schedules used by these groups are often created manually. This process can be complicated and time consuming. As the complexity of the schedule increases, the difficulty of creating the schedule increases exponentially. A tool that is capable of offering several solutions for a user to choose from could also prove to be helpful.

Moving a significant portion of the work of scheduling from a manual process to one that is solved by a computer is the inspiration for Kairos. Kairos can save users a lot of time. Kairos will provide a better solution for those who need to schedule events with even a small amount of complexity.

Since scheduling is such a common problem, Kairos will be attractive to a large number of users. By implementing a well-documented API we intend to attract those who seek a solution that they can tweak to fit their specific needs.

A specific use case that we have identified is the School of Computing at the University of Utah. We are working with staff in the department to provide a solution to their problem of scheduling classes each semester.

2.1.1 Similar Ideas

We have identified two software systems that address the scheduling problem in some manner similar to that which we are aiming at. These systems are Aurora Intelligent Planning and Scheduling System (Aurora) and Microsoft Project (MS Project). To be sure, there are other software systems out there, but these are adequate to represent the current state of this space.

2.1.2 How Kairos is Different

Based on their literature, Aurora addresses the needs that we are attempting to address more than adequately. However, they are focused on supplying solutions to very large organizations with correspondingly very large scheduling problems. We would like to address the needs of the smaller users for which the Aurora software would be overkill.

While MS Project does include scheduling tools in the software package, the larger goal of MS Project is to provide project management software. Thus, their software is attempting to address a problem with a wider scope than we are focused on. Based on reading the literature on their software, it is not clear that the scheduling tools provided in MS Project are constraint-based scheduling tools.

Both Aurora and MS Project cost money. Our software will be free. Neither Aurora nor MS Project are web-based, but require installation on any machine that will use the software. Aurora offers users a packaged software system customized for a given user. By offering an exposed API we will allow smaller users to create their own customizations.

2.2 Required Technology

Our project can be broken down into three main components: a core web service, specific modules that connect to it, and an API that handles communication between the two. We will discuss the technologies we plan to utilize in developing each of these three components.

2.2.1 Core Web Service

This is where the logic for the schedule solver will live. This web service will be centered around the transfer of data through requests and responses. Our web service will use API keys to restrict access. In order to authenticate users, we will store these keys in a MySQL database. This will make our web service more secure.

2.2.2 SoC Module

We will create a module for the School of Computing that connects to our web service. This module will be a web application that allows the user to specify the events (classes) to be scheduled, the resources to be used (professors, rooms, etc.), and the constraints for both classes and resources. When the web service sends back a suggested schedule, the user will be provided with different options for viewing the data. We also plan to provide some web scraping tools in order to facilitate the collection of data relevant to scheduling classes.

This module will have a PHP backend, along with a MySQL database. The front end will be built using HTML, JavaScript, and CSS. We will utilize jQuery (specifically ajax) to make asynchronous requests to the web service without interrupting the user experience.

2.2.3 API

Our API will use JSON to represent the data being shuttled back and forth between the web service and the modules which connect to it. We plan to provide detailed documentation that will help other developers leverage our service in their own applications. To make this documentation readily available and accessible, we will create a website where we will publish the API documentation, and also advertise our service. This site will also be where developers can request an API key.

2.3 Assets and Engines

The websites we create – both the API documentation site and the School of Computing module – will be built using the Laravel Artisan PHP framework. This will provide structure to our web sites, as well as allow us to keep our code organized and concise. Laravel also includes a command line interface which will provide us with shortcuts for common tasks, thus allowing us to focus our efforts on more important development tasks.

2.4 Software/Hardware Requirements for Users

Since Kairos will be a web application with all of the intensive computation taking place on our network, the only system requirements for users will be a machine running a modern browser with a reasonably fast internet connection.

3 Requirements Analysis

3.1 System Architecture

3.1.1 Constraint Solver

At the core of our system will be the constraint solver. The constraint solver is the software that solves general constraint-based scheduling problems. The solver will receive the parameters of a problem instance from the app server and will return a solution back to the app server if one is discovered. The solver will be written in Java and will make use of one or more constraint programming libraries.

A problem instance will be supplied to the solver as a set of activities, a set of resources, a set of constraints, and some objective function. The solver will attempt to fulfill the objective function given the resources and constraints and, if successful, will return a solution to the app server. If it is not possible to fulfill the objective function, a message indicating such will be returned.

3.1.2 App Server

The app server will be the means by which the constraint solver communicates with its users. These users will be one of two types. The first type is the API user. API users will use the solver to solve general scheduling problems. They are converting their concrete scheduling problems into instances of general constraint problems and providing those problem details to the solver. When they receive a solution they will convert it back in order to apply it to their specific scheduling problem.

The second potential user of the app server will be our own Apache server. Similarly to the API user, the problem will be supplied from the Apache server as a general constraint problem. The difference is that these problems will be generated from our implementation of the class scheduler for University of Utah classes.

3.1.3 Apache Server

The Apache server will serve our API documentation website as well as the University of Utah class scheduler site, and the School of Computing class scheduler tool. The API documentation site will be the reference for our API users. It will include all of documentation necessary in order for API users to leverage our solver. It will provide detailed explanations of the methods available as well as code examples.

The University of Utah class scheduler site will presently be a site that will be set up such that departments at the University will be easily able to begin using our scheduling tool. Additionally, it will be extended to fit the specific needs of our primary test user, the School of Computing.

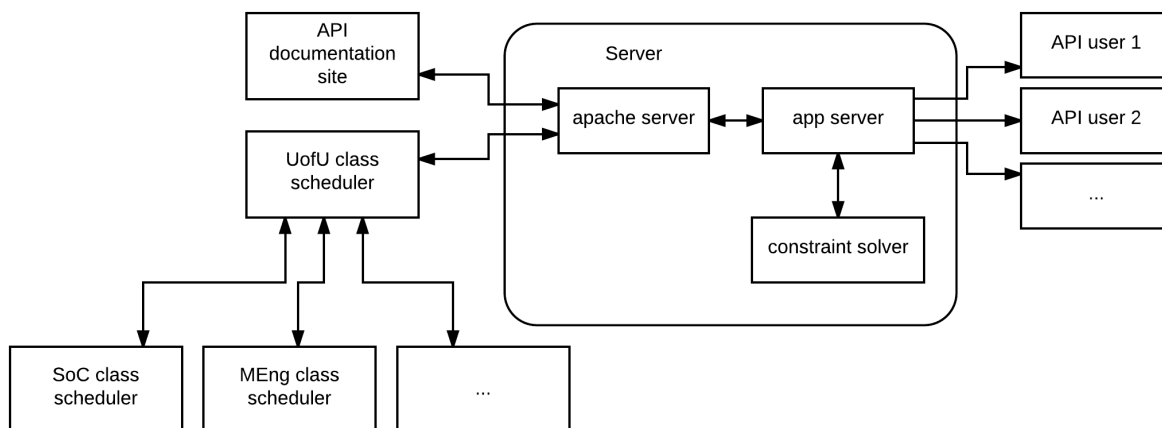


Figure 1: Overview of the System

3.2 Personnel

We have five team members that will work together to implement this project in its entirety. The members are Tyler Chapman, Nate Crandall, Vinh Dang, Vince Oveson, and Tony Tuttle.

Tyler’s primary responsibilities will be ensuring that the network functions correctly and securely. As an extension to his responsibilities to manage the network, Tyler will be the primary person responsible for implementing the server back-end.

Nate’s primary responsibility will be to develop the visualization component of the project. He will be working closely with the front-end developer and will use his knowledge of visualization to make sure that our the user interface will maximize usability and aesthetic appeal. He will also be responsible for designing and implementing the necessary databases.

Vinh’s main responsibility will be implementing tools that are specific to the School of Computing portion of this project. They will need some additional tools such as web scrapers and other peripheral software in order to meet their needs. Vinh will also assist in writing the constraint solver.

Vince’s primary responsibility will be designing the UIs and front-end development. He will use his extensive background in web applications to make sure that all of the front-end components function correctly and provide a pleasant experience for the user. He will also be responsible for design and testing of the overall system architecture, ensuring that all of the parts are working together correctly.

Finally, Tony’s main responsibility will be designing and writing the constraint solver. He will use his knowledge and interest in algorithms and AI to research the best solution and implement it. As a natural extension to this responsibility, he will also be writing the API documentation.

3.3 System Features

Each piece of functionality offered by the Kairos system will be part of one of three levels: basic, planned, and advanced. All of the features in the basic level will be sufficient for a user to use the tool at a fundamental level. Once these are implemented we will focus on implementing all of the planned features. These features will provide the functionality that a typical user might expect. Finally, advanced features are those which we would ideally like to implement to make the system more appealing, but that are lower priority.

3.3.1 Basic Features

The basic features we will implement are:

- *create a new schedule manually* – This feature enables a user to enter each of the classes that need to be scheduled, the rooms and professors available for the classes, and any constraints (such as when a class can be taught, who can teach a class, any two classes that cannot be taught at the same time). These details will be converted to a general constraint problem, fed to the constraint solver. A solution, or a message indicating that none exist, will be returned to the user.
- *weekly schedule view* – This will be the default visualization that we offer to our users. When the system returns a schedule to the user, this is what it will look like. It will display an entire weekly schedule on a single page.
- *auto-save* – Auto-save runs in the background for all Kairos sessions. This feature allows users to make modifications without having to intermittently save session data. This will allow users to try different schedule configurations and commit only when the changes best suit their needs. Auto-save supports our larger goal of minimizing clutter and abstracting details away from the user.
- *user login* – Kairos will provide secure login and registration for users. Providing users with a persistent user id will help secure data and track schedules. It will also allow users to access schedules they have previously worked on and allow them to share schedules with other users.
- *constraint modification (editing a schedule)* – Users will be able to modify a proposed schedule by adding, removing, or editing classes to be scheduled, available professors and rooms, or any constraints on these resources.
- *API* – Our API will be accessible through a public website. Those who wish to use the API to solve their own scheduling needs may do so by acquiring an API key. The API will include detailed explanations about how a user may convert a specific scheduling problem into one that our solver will be able to read, understand, and solve. We will include many code examples to help users get started quickly and easily.

3.3.2 Planned Features

The planned features we will implement are:

- *create a new schedule with CSV* – Users will be able to upload their classes, professors, and classrooms from a CSV file formatted in a manner that we will specify. This will be an alternative to manually entering this data through the web interface.
- *clone a schedule* – Users will be able to create a new schedule by copying (cloning) an existing schedule. Schedules are frequently cyclical with many similarities between schedules at the same point in distinct cycles. This feature allows users to take advantage of this fact and still be able to modify the newly cloned schedule to account for any cycle-over-cycle changes.
- *compare two schedules* – Users will be able to compare two different schedules and identify the items wherein the schedules differ. The two schedules being compared will be displayed as a single schedule where they are identical, with events highlighted where there are differences between the two schedules.
- *drag and drop modification of a schedule* – Users will be able to interact with and modify a schedule by dragging visual objects to desired areas.
- *view schedule by day* – Users will be able to make a selection that will display a single day of a weekly schedule. Doing so will display details that are not visible on the weekly view.
- *view schedule by room* – Users will be able to see all of the classrooms and the classes scheduled for each room.

- *view a subset of classes* – Users will be able to select a subset of all classes that make up the current schedule and view only these classes. This feature may be combined with other view features such as viewing a single day or viewing a schedule by room.
- *import a schedule via CSV* – Users will be able to upload an entire schedule which consists of classes, days and times, professors, and rooms. This will allow users to save locally any of their schedules in CSV format and have the option of importing that scheduling back into Kairos at any time they may need to display the schedule.
- *support multiple users viewing a schedule* – Schedule’s owner will be able to add other users to the schedule. Added users will see the schedule in their dashboard and will be able to view it normally.
- *web-scraping tools* – This will allow users seeing enrollment information for current and previous class schedules. The data will be pulled off automatically from the university’s registration website.

3.3.3 Advanced Features

Advanced features will be implemented after we complete all of the basic and planned features. This set of features represents those items which we would like to implement but will only do so as time and resources permit. The advanced features are:

- *multi-user schedule editing* – This feature allows multiple users to edit a schedule. Users will be able to see the changes made to a schedule by other users.
- *syncing and pulling data from astra.utah.edu* – This will allow users to get available room information directly from <http://astra.utah.edu> instead of inputting it manually.
- *column mapping for CSV import* – This will augment our CSV import functionality by not requiring the user to follow a strict format for their CSV files. Users will be able to name their columns in a way that makes sense to them. When they upload, they will be asked to map their columns to the columns defined by our system.
- *email notifications* – Those parties who want to monitor a specific class for certain changes will be able to request email notifications when such events occur. The emails would be sent, for example, when given classes are moved, or enrollment caps are reached or changed.
- *ability for professors to modify certain settings* – Professors will be able to indicate their sabbatical status and/or the dates/times they are available or prefer to teach. Schedule’s admin can use this information directly for the constraint part when creating a new schedule.
- *multi-user ticketing system* – Non-admin users (ie. professors) would be able to create a support ticket if something on the schedule doesn’t work for them. The schedule admin would be responsible for resolving tickets.
- *professor checker* – This feature will allow users to identify any professors who should have a class assigned to them but currently do not.

4 Tools and Techniques

As a team, we will use the following software engineering tools and techniques.

- *agile vs traditional development* – Our team will use a mixture of both agile and traditional (waterfall) development. We recognize that a good final product requires a lot of careful thought and planning before any coding takes place. Our plan is to follow the waterfall development pattern to map out our requirements and design most of our important features. After this is complete we will switch to a more agile method for the remainder of the development process.

- *version control* – We will use Git for our version control system. We will have a separate repository for each of the main components of our software. Each repository will use two remote repositories: a master hosted on GitHub, and a deployment hosted on our virtual box. We will use git by following a standard workflow procedure:
 1. Prior to beginning work on a new feature or a bug fix, the developer will create a branch from the master branch of the repository.
 2. While working on the feature or bug fix, the developer will make regular commits to that new branch.
 3. When the feature/bug fix is finished and has been thoroughly tested, the developer will notify the team that the branch is ready to be merged back into the master branch.
 4. After at least one other developer has reviewed and approved the changes made in the branch, it will be merged back into the master branch. Conflicts will be resolved as needed.
 5. When features are ready to "go live" we will push those changes to the deployment repository, which will publish the changes to the live website.
- *bug tracking* – For tracking bugs we will take advantage of GitHub's integrated issue tracker. This will help us keep track of known issues, and give us a place to discuss these issues and monitor our progress on resolving them.
- *code review* – New code will not be checked into the master Git branch until:
 1. The code has been thoroughly tested and has passed all unit tests.
 2. The code has been reviewed by at least one other person aside from the primary developer.

Major changes and/or features may warrant a more in-depth code review. These will be held on an as-needed basis.
- *testing* – Testing will be addressed at two levels. Each team member will write unit tests for any code that they write. At the holistic level we will use selenium to test our application. The selenium tests will be designed to fail until all required pieces are implemented.
- *documentation* – We will use Javadoc to document any and all of the code written in Java. The remainder of the code base (PHP, JavaScript, etc) will be documented in a style similar to that of Javadoc.
- *team communication* – For team communication we are employing a combination of regular in-person meetings and electronic communication. We meet as a team at least twice per week and more often as necessary. Between meetings we communicate using a Google group, Google drive, email, and a Trello page. This combination of tools allows us to have extended discussions via a message board, to assign, complete, and keep track of tasks, and to share files.
- *team meetings* – We hold regular team meetings on Monday mornings. These meetings are designed to discuss upcoming obligations, figure out how we will approach the tasks, assign tasks to individual team members, and to discuss any other matters. We also hold a meeting on Wednesdays, immediately after our meeting with the course staff. This meeting is designed to be a debriefing session for the staff meeting, and to check-in with progress since the Monday meeting. We hold other planning meetings as necessary and have also discussed holding work meetings, either in person or via technology, in order to boost our productivity. Aligning schedules has proven to be a challenge for scheduling working meetings thus far.

5 Time Line

Alpha Phase

	Tyler	Nate	Vinh	Vince	Tony
Jan 18	database update, constraint model	drag and drop visualization	constraint model resource gathering	user profile, additional UI components	constraint model resource gathering
Jan 25	configure SSL	drag and drop visualization	implement schedule checking	visualization, UI integration	implement schedule checking
Feb 1	server communication, basic	drag and drop adding classes	implement schedule checking	visualization, UI integration	implement schedule checking
Feb 8	server communication, data parsing	drag and drop server tie in	implement schedule checking	drag and drop server tie in	implement schedule checking
Feb 15	server communication, tie in with solver	server tie in, color scheme design	implement schedule checking	front-end testing	implement schedule checking

Beta Phase

	Tyler	Nate	Vinh	Vince	Tony
Feb 22	branching schedule, multi user support	additional views	implement general solver	CSV data import	implement general solver
Mar 1	ticketing system	additional views	implement general solver	manual data entry	implement general solver
Mar 8	ticketing system, email system	visualization for additional planned views	implement general solver	integrate UI for additional planned views	implement general solver
Mar 15	server communication updates	visualization element builder	implement general solver	UI for visualization element builder	implement general solver
Mar 29	testing	visualization element builder	implement general solver	UI for visualization element builder, create email templates	implement general solver

Production Phase

	Tyler	Nate	Vinh	Vince	Tony
Apr 5	back end for compare schedules	compare schedule visualization	web scraping tool for enrollment data	UI component for web scraping tools	API documentation site
Apr 12	back end for compare schedules	research, implement additional visualizations	web scraping for professor data	UI component for schedule comparison	API documentation site
Apr 19	export tools	testing and bug fixes	testing and bug fixes	UI component for schedule comparison	API documentation site
Apr 26	export tools	testing and bug fixes	testing and bug fixes	help and tutorial for users	API documentation site
May 3	testing and bug fixes	testing and bug fixes	testing and bug fixes	testing and bug fixes	testing and bug fixes

Appendix A Use Cases

A.1 Entering Data Manually to Create a New Schedule

The user will manually enter the data required for the system to generate a proposed schedule. The steps for this use case are:

1. User navigates to Manual Data Entry page
2. System prompts user to enter classes
3. User enters classes and selects 'Next'
4. System prompts user to enter rooms
5. User enters rooms and selects 'Next'
6. System prompts user to enter professors
7. User enters professors and selects 'Next'
8. System prompts user to enter constraints on room availabilities
9. User enters constraints on room availabilities and selects 'Next'
10. System prompts user to enter constraints on professor availabilities
11. User enters constraints on professor availabilities and selects 'Create Schedule'
12. System calculates and returns a proposed schedule to user

See the related Figure 4 on page 15

A.2 Uploading a CSV File to Create a New Schedule

The user will generate a proposed schedule by uploading the required data via a .csv file. The steps for this use case are:

1. User prepares spreadsheet for upload
2. User navigates to CSV Import page
3. System prompts user to enter path to file for upload
4. User enters path and selects 'Upload'
5. System attempts to parse scheduling data from file
 - (a) If the parse fails
 - i. Failure is minimal
 - A. System prompts user to adjust data
 - B. User adjusts data
 - C. Return to [5] above
 - ii. Failure is large
 - A. System informs user of error
 - B. System returns user to Dashboard page
6. System calculates and returns a proposed schedule to user

A.3 Exporting Schedule as a CSV File

The user will download a local copy of a proposed or modified schedule as a .csv file. The steps for this use case are:

1. User navigates to one of the View pages on an existing schedule
2. User clicks Export to CSV option
3. Browser prompts user to accept download
4. Browser begins download of CSV file

A.4 Modifying a Proposed Schedule

The user will make modifications to an existing schedule. They will change the parameters of a given class in order to change some feature of the class. The steps for this use case are:

1. User navigates to one of the View pages on an existing schedule
2. User right clicks on an instance of a class and selects 'Modify This Class'
3. System displays the resources and constraints for that class and prompts user to modify as needed
4. User modifies the resources/constraints and selects 'Modify Schedule'
5. System calculates and returns a proposed schedule with the modifications to user

See the related Figure 4 on page 15 and Figure 5 on page 16

A.5 Comparing Two Schedules

The user will select two existing schedules for comparison. The system will display the discrepancies between the two schedules. The steps for this use case are:

1. User creates and saves at least two schedules
2. User navigates to the dashboard page
3. User selects 'Compare Two Schedules'
4. System prompts user to select two schedules for comparison
5. User selects two schedules and selects 'Compare'
6. System calculates the differences between the schedules and displays them simultaneously, highlighting the differences

A.6 Administrative Tasks

The schedule administrator will be able to load, delete, rename, update users who can view the schedule, and accept/reject changes made by other users of the schedule.

1. User creates and saves a schedule
2. User logs in to the system
3. User goes to the dashboard
4. User selects the schedule(s) that need to be updated.
5. The user can then select their administrative task(i.e. load, delete) on the right sidebar

A.7 API use

The third party developer will request a key and create their own implementation of a constraint based scheduling system using our api.

1. The developer will request an api key
2. The developer will read the api website to learn how to use our api
3. The developer will send the request he needs
4. The system will return json data that the developer can parse and use

Appendix B UI Sketches

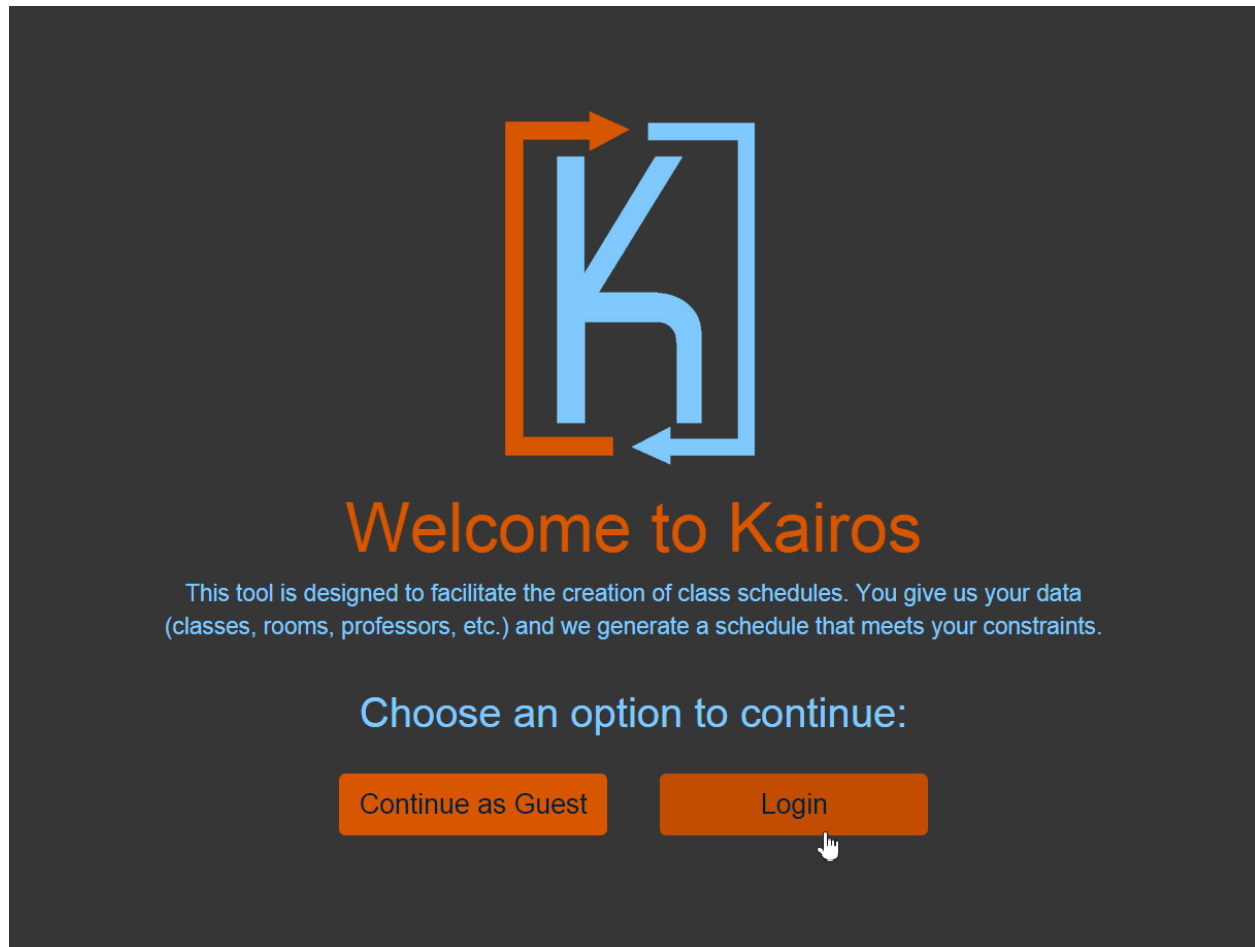



Figure 2: Kairos landing screen


[Home](#)
[Schedules](#)
[Data Tools](#)
[Settings](#)
[Logout](#)

Schedule Admin


Choose a schedule to see details. From there, you can add/remove users, view, edit, copy, or delete the schedule.

New Schedule

[+ Create](#)
[Import](#)

Compare Schedules

Use this tool if you need to see the differences between two of your saved schedules.



[Compare](#)


Schedules

Sort by: Recently Edited


Name	Last Edited	Edited By
Spring 2011	October 31, 2014, 5:44 am	Kelly Olson
Summer 2011	October 31, 2014, 5:44 am	Kelly Olson
Fall 2011	October 31, 2014, 5:44 am	Joe Zachary
Spring 2012	October 31, 2014, 5:44 am	Jim de St. Germain
Summer 2012	October 31, 2014, 5:44 am	Joe Zachary
Fall 2012	October 31, 2014, 5:44 am	Kelly Olson
Spring 2013	October 31, 2014, 5:44 am	Joe Zachary
Summer 2013	October 31, 2014, 5:44 am	Kelly Olson
Fall 2013	October 31, 2014, 5:44 am	Joe Zachary
Spring 2014	October 31, 2014, 5:44 am	Jim de St. Germain
Summer 2014	October 31, 2014, 5:44 am	Kelly Olson
Fall 2014	October 31, 2014, 5:44 am	Kelly Olson
Spring 2015	October 31, 2014, 5:44 am	Joe Zachary


Spring 2014



[Description](#)
[Add a description +](#)



Actions

[VIEW](#)
[EDIT](#)
[COPY](#)
[DELETE](#)



Users

 Kelly Olson

 Joe Zachary

 Jim de St. Germain

[Add a user +](#)


Proposed Changes

[localhost/kairos_ui/public/dashboard#](#)
© 2014 Team Kairos

Figure 3: User schedule selection dashboard

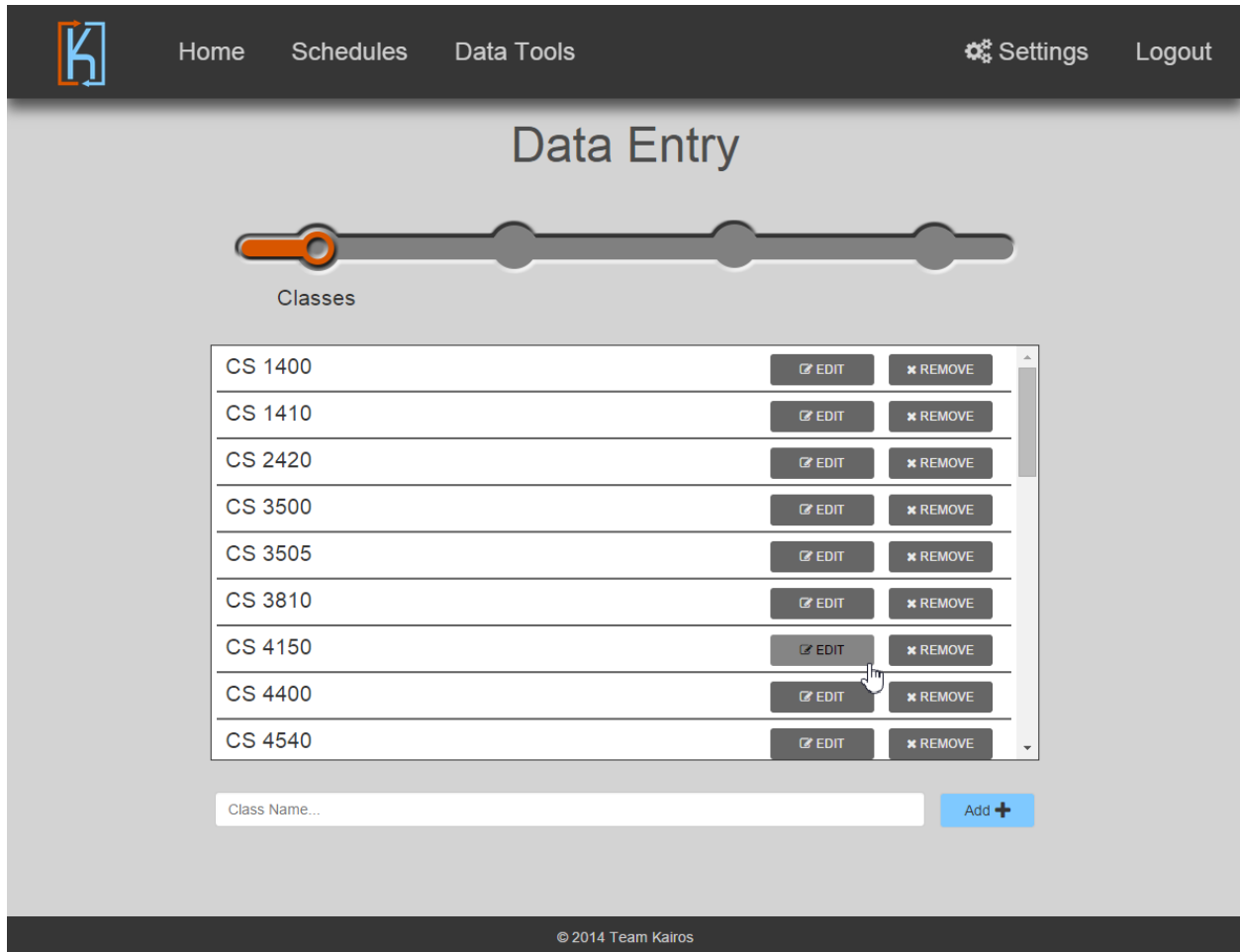


Figure 4: User data entry screen for classes

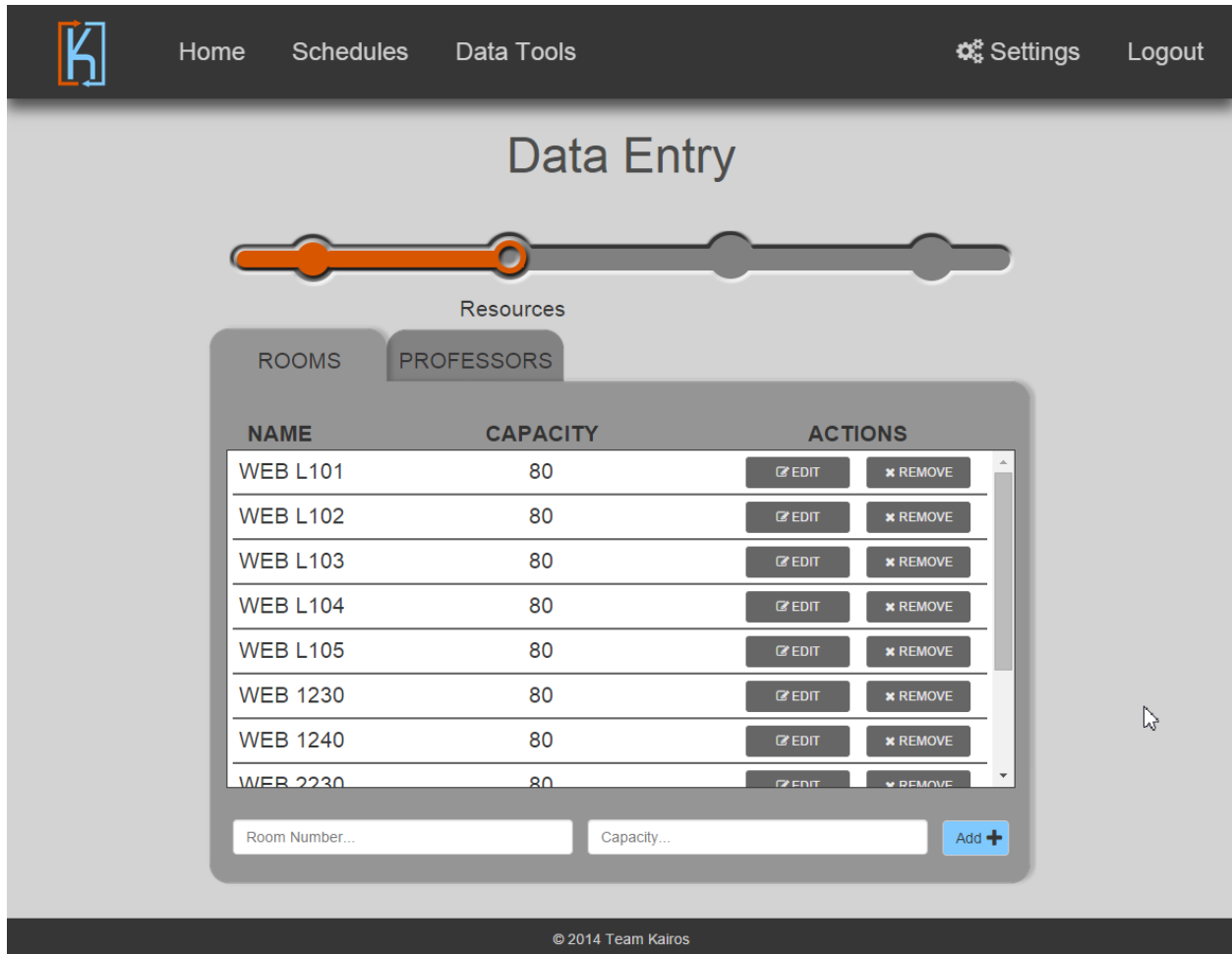


Figure 5: User data entry screen for rooms

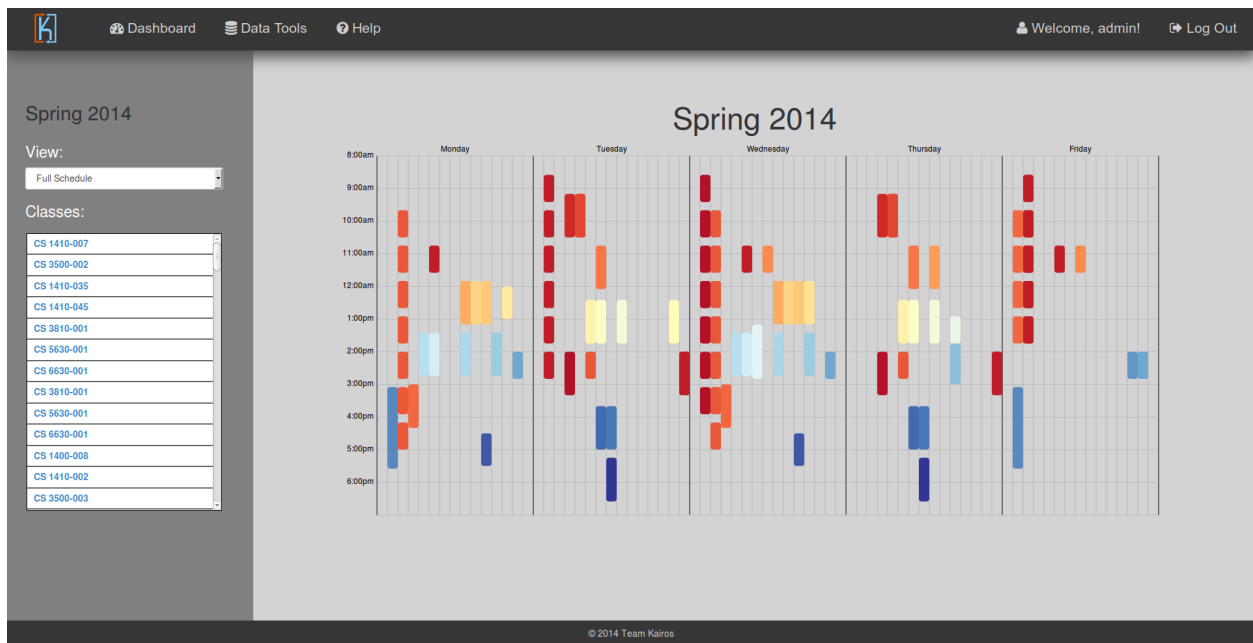


Figure 6: Visualization of a schedule

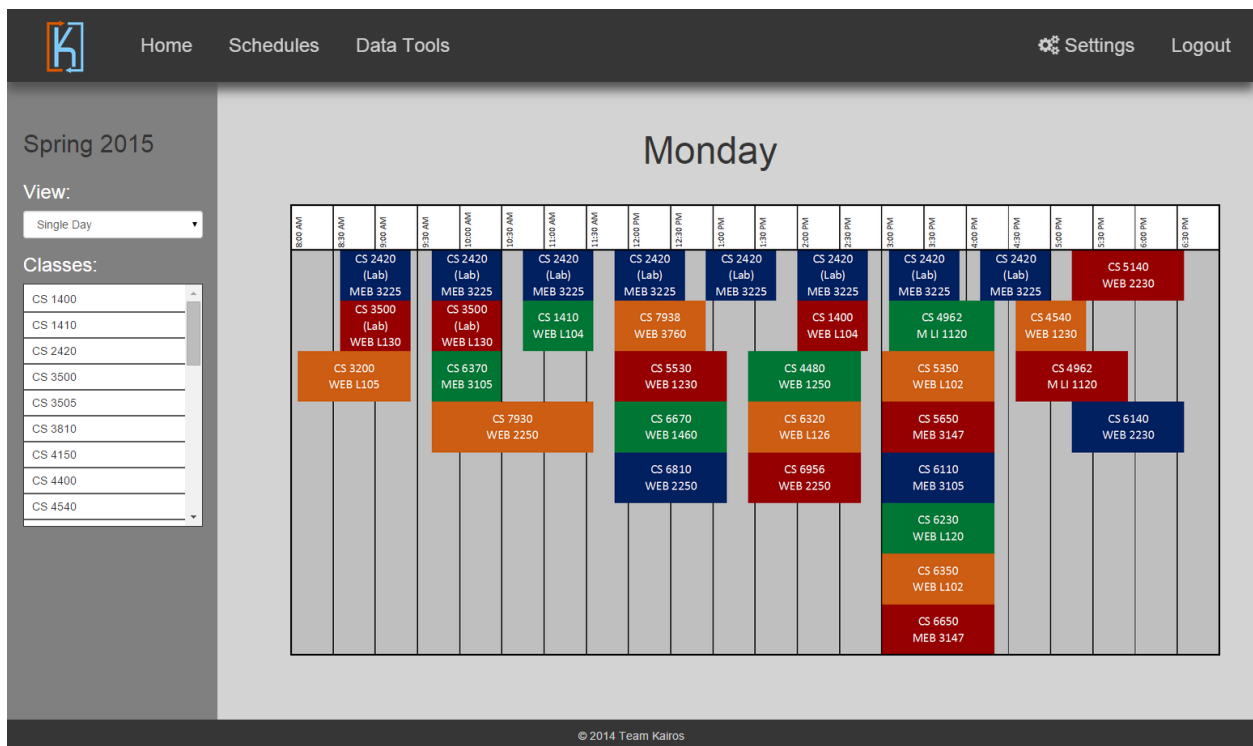


Figure 7: Single day visualization of a schedule