Revision Planner Cross-Platform Web App

Georgi Koshov

Undergraduate Dissertation
Artificial Intelligence and Computer Science

School of Informatics
University Edinburgh
March 2014

THE UNIVERSITY
of EDINBURGH

Abstract

The aim of the project is to develop a working software that has the ability to generate revision schedule for a given list examinable subjects and the corresponding dates. In order to validate the correctness and completeness of the solution, the author has developed a professional-grade web application that has been tested with real users in order to evaluate the qualities, potential and usability of the application.

The application is available at:
http://smartrevise.herokuapp.com/

Table of Contents

Chapter 1

Introduction

- explain the problem and justify the need for the app - explain things such as your desire for a portable tool

1.1 Application overview

The objective of this project is to create an application that allows students to enter a list of subjects and their corresponding examination dates, and produce a revision schedule that fits the following criteria:

- The available revision time is split proportionally between all subjects
- The subtasks of a subject are scheduled in order
- No revision is scheduled outside user-definable working hours
- No revision is scheduled during examinations
- ...

Additionally users have the option to enter periods when they are not available or change subject priorities and receive immediate feedback.

There are several key problems that need to be solved in order to successfully achieve the conceptual aims of the project and to ensure that the minimal viable product could potentially survive and be successful in a commercial setting.

1.2 Application portability

the application should be accessible from a wide range of devices with varying screen-sizes and computational resources,

With the introduction of powerful portable devices that achieve desktop-grade performance in terms of web browsing, it is no longer safe to make assumptions about the screen size or the browser of a given user. Therefore the design of SmartRevise has to be very agile and flexible: the interface elements need to have adequate size in case the user is navigating via touch, the interface has to be fluid, so that it makes the best of both desktop and mobile screens, the technology behind the application should use well adopted web-standards that have been integrated in the majority of modern browsers.

1.3 Solving the scheduling problem

Arguably the most critical task for successful completion of the project is developing a correct and efficient scheduling algorithm that is compact enough to be understood and discussed in its entirety. One might be tempted to implement a full-blown constraint satisfaction framework that allows the user to define sets of constraints imposing conditions on variables. A number of search techniques such as backtracking or local search could be used to derive a solution to a system of constraints on a finite domain. However I will argue in Section 3.2 that such solution is not required to successfully solve the problem as defined in this report. Furthermore constraint satisfaction algorithms are inherently incomplete, that is an algorithm may solve a problem or prove that it is unsatisfiable, but there are some exceptions when a solution can never be derived. Therefore I have decided to focus on user experience, responsiveness and managed to develop an algorithm that is a fraction of the size of a traditional constraint satisfaction toolkit and is provably correct.

1.4 Integration with existing organisation tools

A lot of students use free tools such as Google Calendar or Apple Calendar to organise their schedules and have the information served to all digital devices they use. Google provides an API for developing applications using Calendar's infrastructure and the set up process required for a user to connect their Google Calendar with an external application is mostly effortless and arguably intuitive. Therefore on of the design goals for SmartRevise would be integration with Google's Calendar API and adding support for exporting the generated calendar events to a User's calendar. Additionally, creating a calendar experience within SmartRevise that resembles Google's will make the user's interaction less difficult and considerably more intuitive.

Chapter 2

Background

- describe existing tools which do similar things you probably want to try and say something more general about a wider range of existing work, rather than your more detailed comments on just a couple of application.

In this chapter a brief outline of the available revision planning tools and technology choices will be given. In section 2.1 we discuss existing software with similar functionality and how that compares to the specification for SmartRevise.

2.1 Existing revision scheduling applications

During the project's planning phase a thorough study of the existing applications with related functionality was conducted. The aim was to set realistic goals and success criteria, identify potential pitfalls and justify the need for such application.

While none of the examined products provided the full set of features - particularly the ability to auto-generate revision schedules - the four applications described below came close to the intended functionality of this project and proveded useful insight during the design phase.

2.1.1 ExamTime

htttp://www.examtime.com/ ExamTime is the simpler of the two applications. After registering with the service, the user is prompted to add subjects to their dashboard. Subjects are in fact collections of mind maps, quizzes, flash cards and notes, organised under topics.

There is a calendar view where the user can manually add calendar events for exams, revision sessions or other activities. The application does not allow for

the automatic generation of a revision schedule and the user has to complete this activity by themselves.

2.1.2 Get Revising

http://www.getrevising.co.uk/ Get Revising offers a larger number of features such as Flashcards, Revision Notes, Quizzes, Mind maps and the ability to share resources between students. Some built-in layouts are provided for common GCSE, A Level and University courses that include exam dates and simple content structure for common subjects.

The user of the study planner feature of Get Revising can set School hours that should be excluded from the revision schedule and the application has the ability to split the time prior to exam dates equally between selected subjects.

However Get Revising does not allow for a deeper level of detail when it comes to creating a schedule - only events for revising a particular subject can be created, therefore a user cannot include items such as*read textbook, solve tutorials, do past papers, etc.*

2.1.3 S-cool

S-cool is primarily targeted at high-school students doing GCSE and A Level courses. The web-site requires visitors to create an account before accessing any of its features, prompting for potentially sensible data such as full name, birth date and postcode. Furthermore, a confirmation e-mail was sent immediately following the registration, that contained a plain text record of the password entered in the website.

Although discussion of the potential security implications caused by these design decisions is beyond the scope of this thesis, security best-practices have been followed while developing this project and further details can be found in section 4.4 Security.

After registering, users can create an exam timetable and manually enter multiple revision periods. The simple month view calendar only allows students to enter the date, but not the exact time of events, rather they are displayed as items in a task-list for a given day.

2.1.4 SQA "My Study Plan" and "My Exams" for iPhone

These two mobile applications were created in conjunction with the Scottish Qualifications Authority to help pupils in secondary school to organise their studies and revision. "My Study Plan" was last updated on 8 April 2011 and

does not appear to be actively developed, while "My Exams" is still maintained. The iPhone app is targeted at students in Scottish secondary school and allows them to create exam timetables, add notes to subject areas and share their timetables.

However, the application lacks integration with desktop tools, while the timetable only allows users to see a list of upcoming exams.

Chapter 3

Scheduling Algorithm Design

- what criteria are you going to allow the user to specify? - how did you decide that these are appropriate? - discussion of PERT etc and why it might not be a particularly fit - mention of constraint-based approaches - description of the algorithm that you finally used

In Section 3.1 the reader may find a high-level summary of existing project-management techniques and some arguments about their applicability in the current project.

Due to the advancement of faster, more powerful web-browsers, featuring robust engines for running efficiently large amounts of code and with the continuous effort by organisations such as Mozilla and Chromium to drive web standards further and develop powerful open-source technologies for the next generation of Web, it is no longer infeasible to develop an entire application stack using a single programming language - ECMAScript, or more commonly known as JavaScript.

Some of the arguments in support of such design decision include the ability to exchange JavaScript objects between server and client without having to convert them during transport, being able to plug community libraries at all levels of the project without having to worry about compatibility or portability, but most of all due to the sheer convenience of having an entire application stack built with the exact same tools and following the same design philosophy.

A more traditional way to solve the problem defined in this report would be to create a server-centric application that relies on technologies such as PHP, Python or Java to do the majority of the work on the server and send the rendered HTML templates as static files to the client. While this solution is widely used in practice, recent years have seen a change in the web application design approach, targeted at thinner back-ends and outsourcing the majority of the application logic to the user's browser.

3.1 Discussion of scheduling methods

Several options for scheduling of concurrent events exist in the project-management domain. Critical Path Method and Program Evaluation and Review Technique are the two most widely used techniques for solving a project-management problem. This section will include a brief overview of the two techniques as well as some discussion of the applicability of CPM and PERT for the current problem.

3.1.1 Critical Path Method

3.1.2 Program Evaluation and Review Technique

3.1.3 Constraint satisfaction

Traditionally CPM and PERT are used when solving problems where a number of resources have to be shared between many possibly concurrent tasks. Example scenarios include organising receptions and events where many workers need to complete several tasks that could include some dependencies.

The problem that SmartRevise is attempting to solve assumes that a single worker - the student - needs to decide on a suitable ordering and allocation of several streams of independent tasks. There is no notion of event overlap - at any given point the student is revising a single subject. Additionally we assume that one subject does not depend on another since all courses are deemed self-contained. Therefore the scheduling algorithm needs to track only dependencies of the form: Read textbook for subject X $\rightarrow$ Solve tutorial questions for subject X $\rightarrow$ Attempt past papers for subject X $\rightarrow$ Do exam of subject X. Therefore neither CPM nor PERT are directly applicable in this case. However, when determining the critical path a typical CPM implementation walks the network diagram of a scheduling problem forwards and backwards. A similar approach will be outlined on Section 3.2 when efficiently deciding on a suitable partitioning of the available time.

3.2 SmartRevise algorithm

The algorithm that was developed for SmartRevise revision scheduling solves the problem on incompleteness, typically associated with constraint satisfaction, by limiting the domain and introducing a single relaxed constraint - amount of hours spent on each subject. The algorithm successfully finds schedules that are intuitively correct and runs efficiently without any noticeable delays. Most of all, the entire scheduling software is executed in the user's browser, which

makes the application self-contained and less reliant on a centralised server with high performance.

The SmartRevise algorithm takes the following input arguments:

Exams
An array of **Exam** objects. Each object has the following attributes:
Title - name of subject
Date - date and hour of examination
Duration - duration of examination
Portion - a floating point number [0,1], the relative portion of revision time that will be spent on subject
Components - sub-tasks that need be completed as part of the revision for the subject

Revision start
Usually the current time

Daily start and end time
Used to define waking hours available for revision, e.g. 9AM - 6PM

Additional blocking events *optional*
Events that indicate a period when the student is not available

The pseudo-code outlined in Listing 3.1 presents the basic structure of the SmartRevise scheduling algorithm:

Listing 3.1: "SmartRevise scheduling algorithm"

```
Exams.sort(exam.date, ascending);
var revisionEnd = Exams[exams.length - 1];
var revisionLength = difference(revisionStart, revisionEnd);

// Divide revision period in "chunks" of time between exams
// Examine days in reverse order starting from last day
for (var day=revisionLength; day>0; day--) {
       if (day.hasNoExams) {
              currentChunk.slices.push( day );
       } else {
              currentChunk.slices.push( period(exam.date+exam.duration,
                  day.end) );
              currentChunk = new Chunk
       }
}

// Calculate total revision time
var revisionTime;
for (chunk in chunks) {
       for (slice in chunk.slices) {
              revisionTime += chunk.length;
       }
}
```

```
// Determine revision time per subject
for (exam in exams) {
       exam.time = revisionTime * exam.portion;
}

// Starting from last chunk proportionally (to remaining hours of each
    subject) split time between exams, keeping track of spent hours
var events = [];
for (chunk in chunks.reverse) {
       for (slice in chunk.slices){
              // Determine the total remaining revision time for the
                  exams remaining after the current chunk
              var chunkExamsTime;
              for (exam in chunk.exams) {
                     chunkExamsTime += exam.time;
              }

              // Allocate proportional amount of time to each exam
              for (exam in chunk.exams) {
                     var subjectTime = (exam.time / chunkExamsTime) *
                         slice.length;
                     exam.time -= subjectTime;
                     // Create new revision event and add it to the list
                         of events
                     events.push(new SubjectRevision(exam.title,
                         subjectTime));
              }
       }
}
```

Each Chunk object contains several Slices - an uninterrupted working time, e.g. 31/06/2014 09:00 - 31/06/2014 18:00. A chunk also has an Exams attribute that contains an array of Exam objects with due date after the end of the chunk. Therefore the first chunk contains all slices from the start of the revision until the first exam date. Additionally $Chunk[0].exams$ contains all exams in the schedule.

Using the logic outlined in the pseudo-code in Listing 3.1 and by including edge-case handling and means for converting and executing mathematic expressions on JavaScript Date objects, a JavaScript method was developed that can successfully partition the period between a given start date and a number of random exam dates.

One notable exception surfaced during the development of the algorithm - if an exam is scheduled at the very beginning of a lengthy exam diet it might be the case that there is not enough time between the revision start and the exam date to spend the total time allocated for that subject. Despite that, such subjects are allocated most of the available time and without adding any specific han-

dling for such cases, the algorithm produces sensible and meaningful revision schedules.

Chapter 4

Architecture and Implementation

- choice of tools and justification - RWD

4.1 Project architecture

SmartRevise is a client-side JavaScript application based on the Google AngularJS framework that is served by a NodeJS server. The entire application is downloaded by the client upon establishing connection with the server and the user's browser is responsible for rendering views and navigating between them. A similar approach is used by Google for their Gmail application and it allows for offloading the rendering responsibility from the server to the client. Additionally, after downloading the application, the only data that is exchanged with the server is in the form of JavaScript objects. Since the front-end can be cached by the browser, this solution can decrease traffic and improve the user's experience. On the other hand, since all further server requests are executed via an API to the database, it is trivial to implement memory caching, load-balancing and advanced protection to the leaner and less complicated back-end.

Application deployment, provisioning of the environment and dependency management is also handled by JavaScript services such as Grunt and Bower. Further details about configuring and using those can be found in section 4.5.

4.2 Client-side application

The front-end of SmartRevise was built using AngularJS. This is a front-end web application framework built by engineers at Google, aiming for rapid development of lightweight yet powerful browser applications. It follows the $Model-View-Controller[MVC]$ philosophy and allows the developer to built modular applications by allowing the abstraction of every single interface element or piece of functionality in an Angular structure called Directive. A di-

rective could be anything from a simple UI element with additional features, to complex scheduling algorithms that can be packaged and distributed with ease.

There are alternatives to Angular such as EmberJS, Knockout and Backbone. The decision to use Angular was made in an effort to extend the author's knowledge of JavaScript MVC frameworks and due to the abundance of resources about the software on the Internet.

4.3 Server and REST API

The back-end of the application was built using NodeJS. This is a JavaScript platform built on Google Chrome's JavaScript V8 engine. It is a lightweight event-driven framework that aids the building of scalable and efficient server application.

Routing, request handling and connection to the database is achieved using Express - a JavaScript framework for building web applications based on Node. The purpose of the library is to provide basic functionality such as authentication, user sessions and request serialisation out of the box.

The database was built using MongoDB. This is a mature open-source NoSQL database written in C++ that provides excellent interfaces for data exchange with Node application. Mongo features an SQL-like query language based on JavaScript and allows the developer to store entire JavaScript objects without having to convert back and forth to a relational schema. NoSQL solutions are not optimal for performance-centric applications or solutions involving massive amounts of data. In the case of SmartRevise there are several arguments in support of such design decision:

Rapid development
: Changes in the database schema are largely effortless since there is no need to convert JavaScript objects to tuples in a relational database

One lookup and update per user session
: Since SmartRevise is a client-side application, all changes to the data are made in the user's browser and stored locally until the end of the interaction. Therefore the number of database lookups is kept minimal

Ability to switch database engine at any point
: The above items outline MongoDB's advantages during development. Since all data calls are routed via an API, it would be trivial to switch MongoDB with a traditional SQL database and develop adapters that convert SQL relations to the corresponding JavaScript objects. Currently there is no need for higher performance, but should any issues emerge that cannot be solved by the MongoDB community, the architecture allows for quick exchange of components.

4.4 Security

TODO Sessions, salted hash passwords, no SSL

4.5 Additional tools

This section will provide a review of the miscellaneous software packages that were used in order to ease development, improve the quality of the software and simulate as closely as possible a real production environment.

Grunt

Grunt is a JavaScript Task Runner. It allows the developer to automate repetitive tasks by defining a Gruntfile.js with automation parameters. The biggest advantage of Grunt over similar technologies such as Puppet or Chef is its growing environment focused on JavaScripts.

In the setup for SmartRevise, Grunt is used to compile multiple JavaScript libraries to a single script file, compress the code using best practices in order to achieve faster and smoother experience for the user, run the database and server applications as well as monitor the project directory for file writes and automatically recompile the application.

Additionally using Grunt one can generate templates for various Angular components such as new routes, templates or directives, as well as export a compiled version of the application that can be published to a SaaS provider such as Heroku in less than a minute.

In summary, Grunt is a crucial component in every professional toolkit nowadays and helps deliver ideas very quickly without technology getting in the way.

Bower

Bower is the first JavaScript package manager for front-end web projects. It makes downloading and managing a large number of external dependencies effortless compared to manual management of libraries.

Package management is traditionally considered a trivial problem for modern software engineering, but since the majority of JavaScript tools are relatively young and less developed, being able to manage the entire stack of a web application with ease and convenience is a significant step forwards that makes this project better suited for future development.

Chapter 5

Evaluation

Evaluating the correctness and completeness of this project involves a two-step process.

Determine corectness of developed algorithm

A number of constraints can be associated with the schedule derived by SmartRevise:

- Revision for a given subject should only occur prior to the examination
- Revision events should not be allocated during sleep hours
- Only one revision event should be scheduled at a given time
- etc.

The application stack used for this project allows the creation of unit tests. It is trivial to generate a set of example subjects and examination dates and test the produced schedule against the constraints listed above. Together with a study of some known edge-cases this can give a sensible evaluation of the correctness of the proposed algorithm.

Evaluate usability of application TODO - RWD tests?

Arguably the biggest challenge is developing a visually appealing, intuitive user interface that suits the needs of test users and can provide some value if added to a student's workflow.

Unfortunately testing the finished application in a real exam setting would be impossible before the deadline of this project, but the author intends to evaluate the application extensively with fellow students in order to identify potential sources of confusion, any shortcomings or inconsistency. However there are few well-studied methods for formally evaluating user experience, therefore the main form of evaluation will be verbal feedback and observation.

Chapter 6

Further work

Integration with existing tools etc.

Chapter 7

Conclusion

As indicated in this report, the author has studied the applications that advertise similar functionality to the one intended by the project specification. Applicable algorithms and scheduling techniques have been studied and significant progress has been achieved in developing a working implementation of the outlined architecture.

Overall the project is on schedule and the application appears to meet the design goals of the given specification.