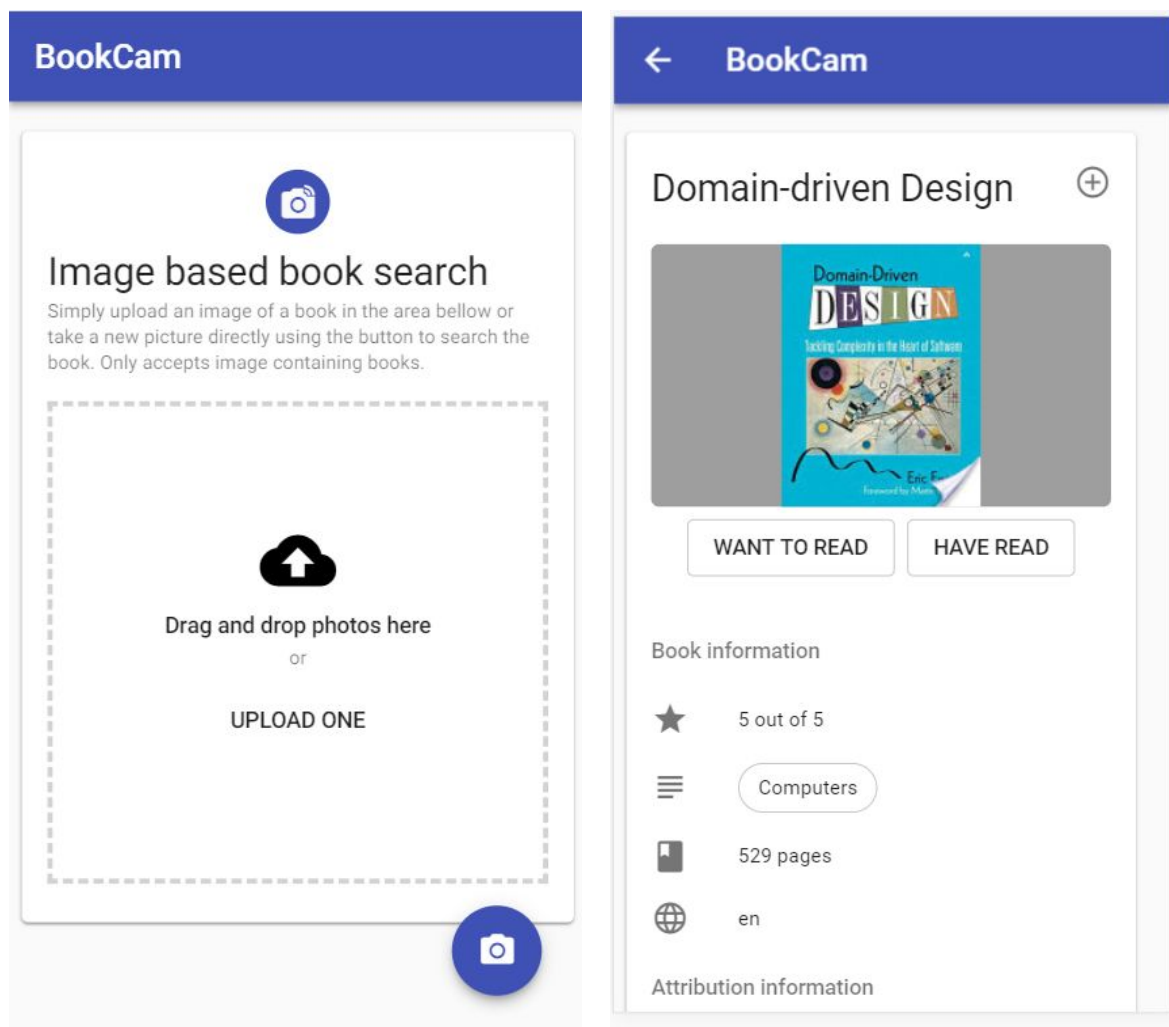


CAB 432 – Cloud Computing

Assignment 1 rapport

By Jordi Smit - 10264139



Demo:

<http://book-cam.com/>

Table of content

1. Introduction	2
1.1 Google Cloud Vision API	2
1.2 Google Books API	2
1.3 Goodreads API	2
1.4 Bol.com API	3
2 User stories	3
2.1 Use case A	3
2.2 Use case B	5
2.3 Use case C	7
3 Technical Breakdown	8
3.1 Client side	8
3.2 Server side	9
3.4 Building tool	9
3.4 Docker	9
4 Issues	10
4.1 Loading time	10
4.2 Goodreads API	11
5 Testing	11
5.1 Error handling	11
5.1.1 Errors during the image search	11
5.1.2 Goodreads cannot find the book	12
5.1.3 No online offers could be found	13
5.1.4 Uncaught error on the client side	13
5.2 test cases	15
6 Extensions	16
6.1 Other users have searched	16
6.2 Additional online vendors	16
6.3 Allow text based searches	16
Bibliography	17
Appendix A Docker deployment	18
Appendix B User guide	19

1. Introduction

The aim of this project is to make it easier for users to manage their digital libraries and find information about the books they own. This mashup combines Google's Vision (1.1), Google Books (1.2), Goodreads (1.3) and Bol.com's API (1.4) into a image based book search. With this project users will be able to perform book search queries using uploaded images or by taking a new picture when they are using the application with their smartphone. The application will verify that the uploaded picture contains a book. When the picture contains a book the user will be provided with all kinds of information about the book like descriptions, authors, reviews, similar books, price, where to buy it, etc. If the user is logged in using his Goodreads account he will be able to automatically add the book to his library shelf or others.

The application has been deployed on:

<http://book-cam.com/>

1.1 Google Cloud Vision API

The Google Books API ("Vision API - Image Content Analysis | Cloud Vision API | Google Cloud," n.d.) is a book search service which provides general book information like descriptions, titles, Authors, etc. This project will make use of the label classification to detect if the user has uploaded a book and uses the text in the image to produce a search query for Google Books. The other available information like faces, landmarks, logos, etc will not be used for this project and thus will not be requested from the API.

1.2 Google Books API

The Google Books API ("Google Books APIs | Google Developers," n.d.) is a book search service which provides book information. This API is able to perform natural language search queries which makes it ideal for the searches based on the raw output of the Google Vision API. This API will allow us to find the book title which can then in turn be used by the other APIs which require title based searches.

1.3 Goodreads API

The Goodreads API ("API," n.d.) provides access to the Goodreads data such as book information, related books, reviews, author information, etc. When a user has logged in into Goodreads using this application the API also allows for user account interaction such as updating user's shelves, create reviews, send friend requests, etc. This project will use this API to retrieve general book information, book reviews and to allow the user to add the book to one of his shelves.

1.4 Bol.com API

Bol.com (“bol.com | de winkel van ons allemaal,” n.d.) is a Dutch online shop similar to Amazon.

The Bol.com API (“Home - Bol.com Developer Center,” n.d.) provides access to Bol.com’s data such as products, prices, stocks, etc. This application will use this API to provide the user with a price indication of the book and webshop links, for when the user does not yet own the book.

2 User stories

2.1 Use case A

As a user I want to be able to find relevant information, related books and reviews from a book that I have seen, by taking a picture of it.

The user navigates to the index page. On this page the user will take a photo or upload a photo of the book in question (Figure 2.1.1).

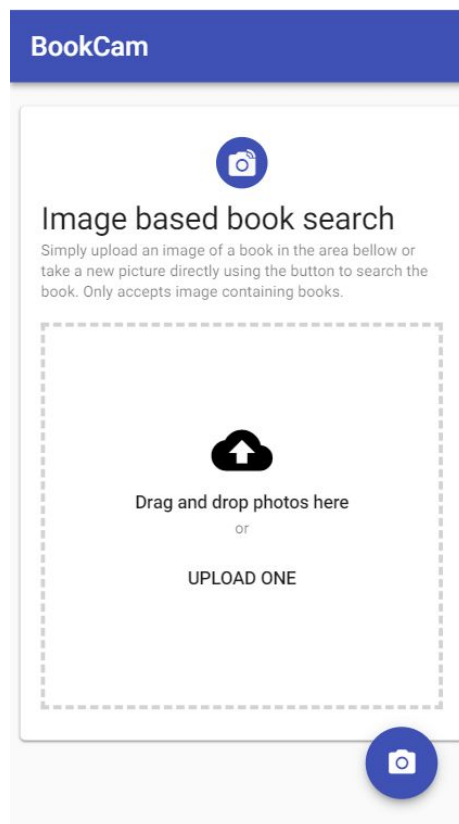


Figure 2.1.1 The index page

If the user has uploaded an image that does not contain a book the user will be shown an error dialog. However if the image contains a book the image will be analysed and an automatic search query will be performed and the user will be shown the 3 most likely books (Figure 2.1.2).

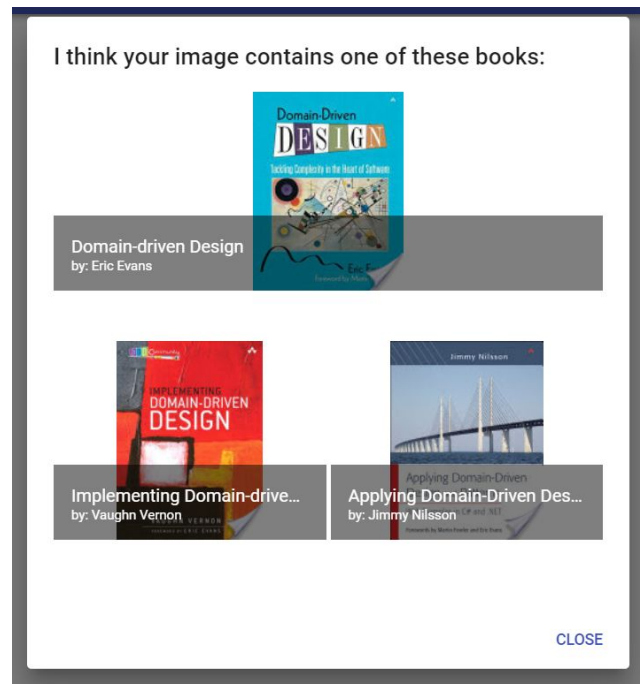


Figure 2.1.2 The search results

The user will be redirected to a dedicated book page when the user click one of these books. On this page the user will be provided with all kinds of information of the book in the image, such as a description, author information, related books, reviews, etc (Figure 2.1.3).

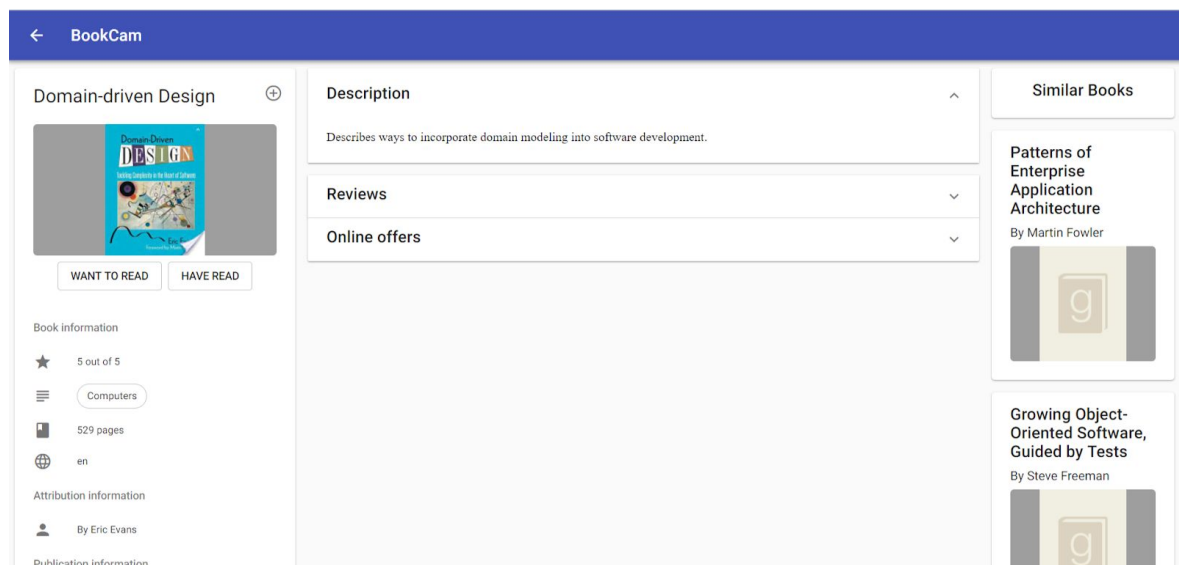


Figure 2.1.3 The book information

This use case uses the following services: Google Vision API to classify the image, Google Books API to find the book information, Bol.com to find the offers and prices.

2.2 Use case B

As a user I want to be able to add a book to one of my Goodreads shelves by simply taking a picture of it.

The user navigates to the index page. On this page the user will take a photo of the book in question. Based on this picture the user will be redirected to the dedicated book page as specified in 2.1. This page contains a button that allows the user to automatically add the book to one of his Goodreads shelves (Figure 2.2.1).

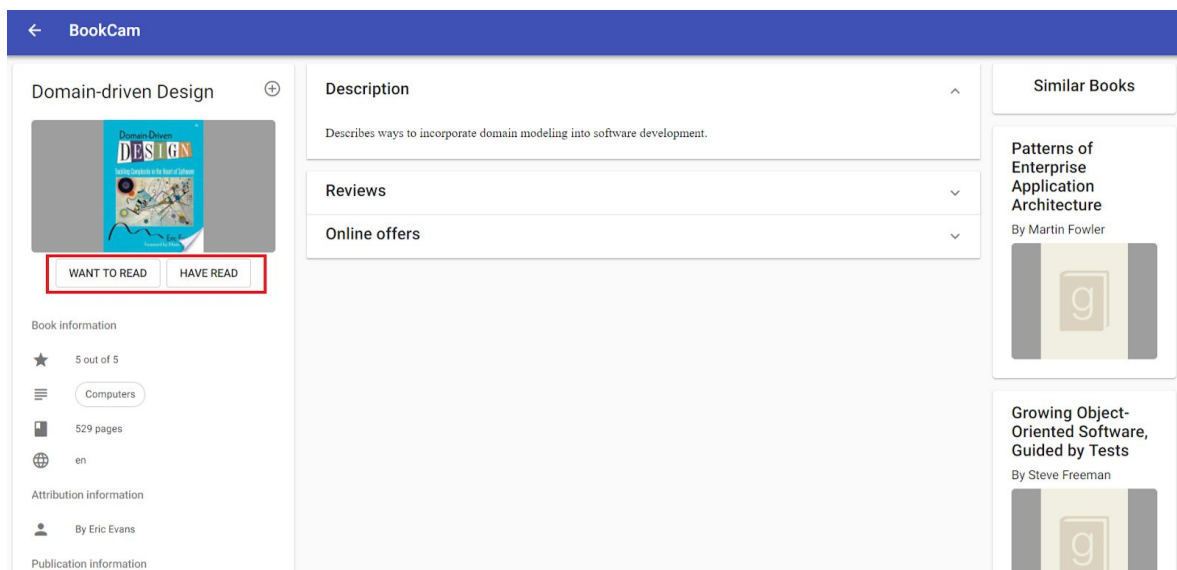


Figure 2.2.1 The book information

After clicking one of these buttons the users will be redirected to the Goodreads login page (Figure 2.2.2).

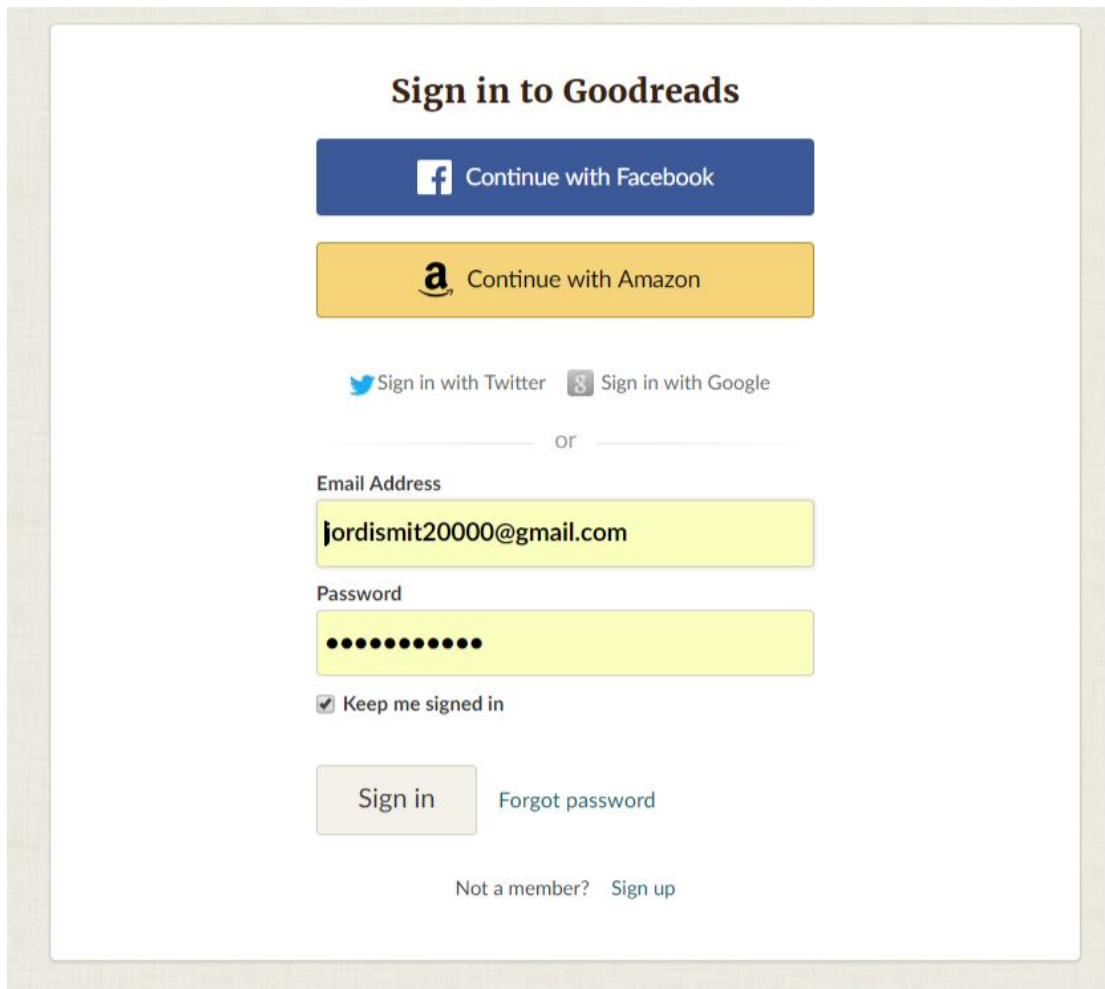


Figure 2.2.2 The Goodreads login page

When the user has logged in, the user will be redirected back to book page on which he started. On this page, the user will be presented with a status message in a Snackbar which will tell his if the book has been added to his shelf or not (Figure 2.2.3).

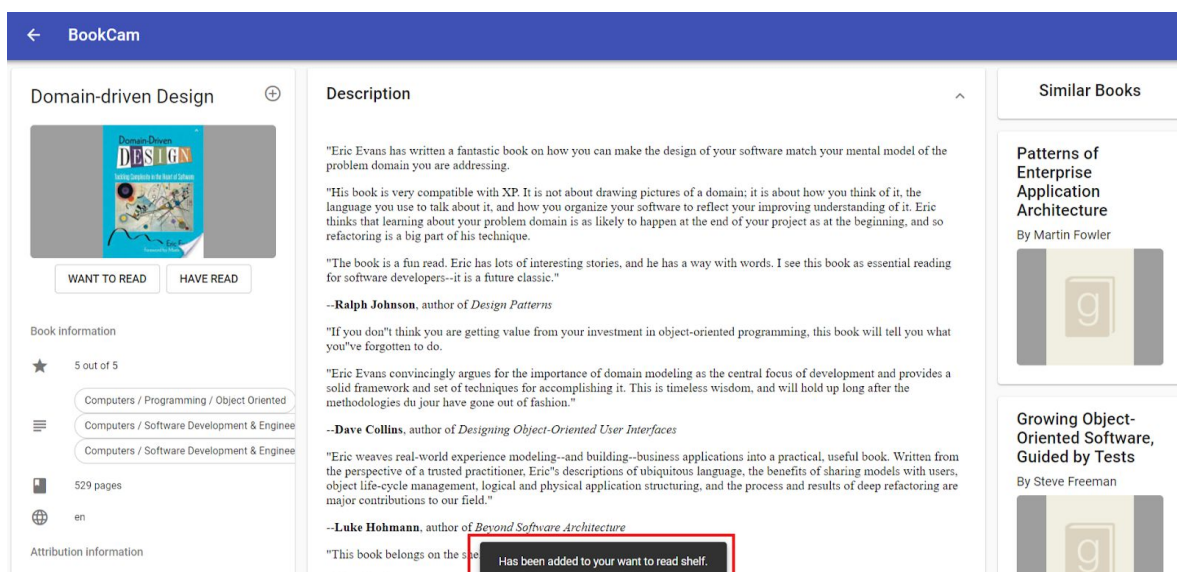


Figure 2.2.3 The book has successfully been added to your shelf message

This use case uses the following services: Google Vision API to classify the image, Google Books API to find the book information, Goodreads API to update the user's shelf.

2.3 Use case C

As a user I want to be able to find online shopping links for a book that I have seen by simply taking a picture of it.

The user navigates to the index page. On this page the user will take a photo of the book in question. Based on this picture the user will be redirected to the dedicated book page as specified in 2.1. This page will provide the user with a price indication of the book. The user will also be provided with the cheapest offers for this book currently on Bol.com (Figure 2.3.1).

The screenshot shows the BookCam app interface. The top bar is blue with a back arrow and the text 'BookCam'. The main content area is divided into several sections. On the left, there is a book cover for 'Domain-driven Design' with a 'WANT TO READ' and 'HAVE READ' button below it. Below the book cover, there is a 'Book information' section with a star rating of 5 out of 5, a category of 'Computers', 529 pages, and the language 'en'. The 'Attribution information' section shows the author 'By Eric Evans'. The 'Publication information' section is also visible. The main content area has a 'Description' section with the text 'Describes ways to incorporate domain modeling into software development.' Below this is a 'Reviews' section. The 'Online offers' section is highlighted with a red border and contains a table with the following data:

Seller	Price	Condition	Availability
bol.com	71.38	Nieuw	available

On the right side of the interface, there is a 'Similar Books' section with two book covers: 'Patterns of Enterprise Application Architecture' by Martin Fowler and 'Growing Object-Oriented Software, Guided by Tests' by Steve Freeman.

Figure 2.3.1 The online offers overview

When the user clicks on one of the offers, the user will be redirected to the corresponding Bol.com page (Figure 2.3.2).

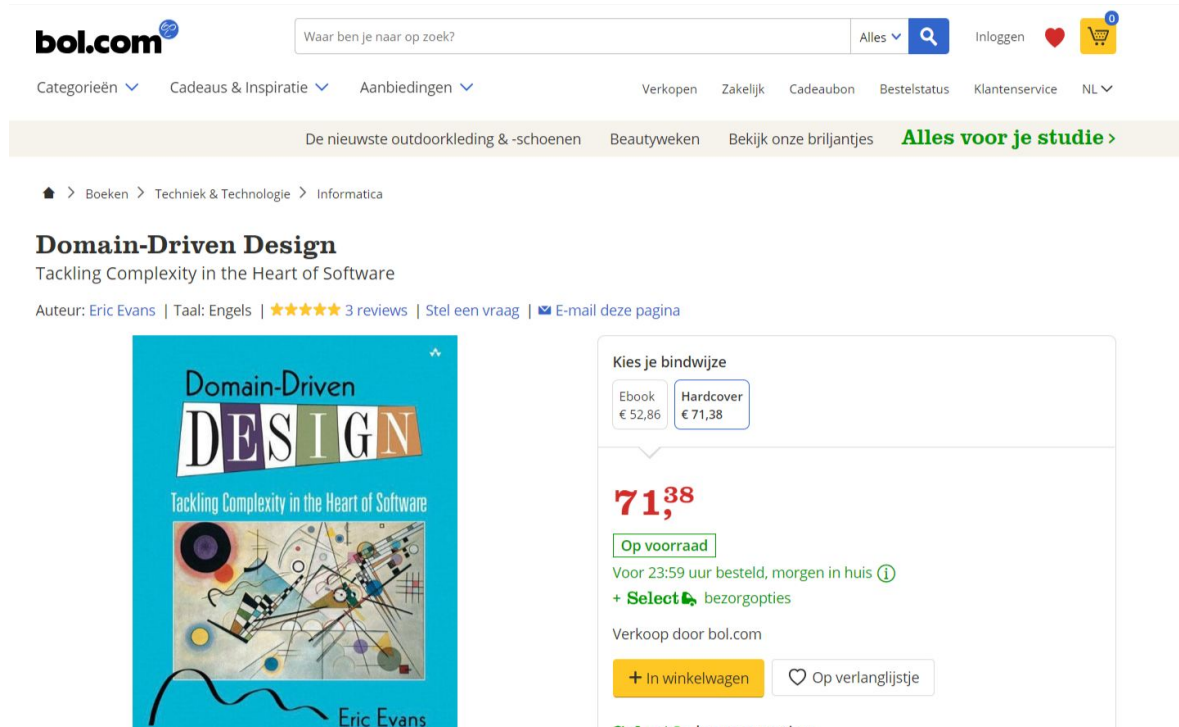


Figure 2.3.2 Redirect result

This use case uses the following services: Google Vision API to classify the image, Google Books API to find the book information, Bol.com to find the offers and prices.

3 Technical Breakdown

The aim during the design phase was to keep the client side processing as minimum as possible. Therefore the client side (3.1) will only be responsible for creating the views, handling the UI events and requesting the combined data from the server end points. While the server (3.2) will be responsible for handling the requests and combining the services of the external APIs. The source code for both client and server side have been built using tools (3.3) such as webpack and typescript and can easily be deployed using a docker image (3.4).

3.1 Client side

The client side has been built using the ReactJs framework ("React – A JavaScript library for building user interfaces," n.d.). This framework allows developers to create interactive UIs without the need to reload the entire page. When the user navigates to the website he will be served with the entry point of the ReactJs application. The React-Router will read the current url and will serve the defined view for this url (or a 404 page). React-Router also allows the user to navigate between urls without reloading the entire page. Each view will request the

data it requires from the server, on a as needed basis, using AJAX to ensure that the application runs as fast as possible.

User inputs from UI events, such as uploading an image or adding a book to a shelf, will also be send to the server using AJAX and based on the response from the server the views will be updated accordingly.

3.2 Server side

The server exposes multiple endpoints using a rest design on the /api path on which the client side can request the data it requires using AJAX. The server composes the requested data on these endpoints by transforming the original request inputs and using them as input for the external APIs. The server then combines the results of these APIs into the requested format and return the result to the client side as JSON.

Google Vision ("npm: @google-cloud/vision," n.d.), Google Books ("npm: google-books-search," n.d.) and Goodreads ("npm: goodreads-api-node," n.d.) all had their own javascript library that abstracted away their REST requests. These libraries have been used for this project to make the access to the external APIs easier. Sadly there was no library available for the Bol.com's API. Thus requests to this API have to be done manually using fetch operations on its REST API.

3.4 Building tool

In order to make the development process of this project easier, Typescript ("TypeScript - JavaScript that scales," n.d.) and Webpack ("webpack," n.d.) have been used as development tools.

Typescript introduces type checking to Javascript making development much less error prone. In this project both the client and server side have been written in Typescript, which resulted into a far less error prone development cycle.

Webpack is a module bundler that combines the source code and transforms it into a lower version of javascript that is available on more devices. In this project Webpack has been mainly use to transform the Typescript source code into Javascript and to bundle the client side source code files into a single bundle.

3.4 Docker

Docker has been used to encapsulate the application into a single image. Which makes the application very easy to deploy using DockerHub. The application's image has been built using the following docker file:

```

FROM node:8
# Create app directory
WORKDIR /usr/src/app
# Install app dependencies
COPY package*.json ./
#Install the runtime and compile dependencies
RUN npm install
# Bundle app source
COPY . .
#Expose the server port
EXPOSE 8080
#Compiles the project.
RUN npm run-script build
#Starts the server.
CMD npm start

```

Figure 3.4.1 The DockerFile

The build extends the official node 8 build which provides us with a linux image that has node already pre-installed. The build then continues by creating a work directory in which it will copy the package files of the application and install the project's dependencies. When the dependencies have been installed the source code will be copied into the image. This step is done after the dependencies installation based on the assumption that the source code will change more often than the dependencies. So when a rebuild is needed due to source code changes the long "RUN npm install" step does not have to be redone. Once the source code has been copied the Typescript code will be compiled into Javascript. And finally the server will be started.

4 Issues

During the project I encountered two problems. The first one was related to the loading time of the book page (4.1) and the second one was related to the Goodreads API (4.2).

4.1 Loading time

The first version of the application combined the data of all the different APIs before sending it to the client side. The idea was to provide the client with all the information he would need in a single request. However with this approach the server had to wait for all the API responses before it could provide the client with any data. The server's response time grew in relationship with the amount of APIs in use. Eventually the response time became over 5 seconds. So I decided to split all the API calls in order to decrease the waiting time for the client. This would allow the user already get started with the data that is already available. However this would also increase the complexity on the client side.

4.2 Goodreads API

The Goodreads API library does not allow us to make request that requires user's authorization using AJAX (such as adding a book to a shelf in our case). The user must first be redirected to the login page of Goodreads before one of these requests can be made. This resulted into a problem, since the entire client side has been written in ReactJs which heavily dependent on AJAX. Sadly I was unable to find a way around this problem using AJAX. So I created a redirect URL that redirects the user to the Goodreads login page before adding a book to the user's shelf. After the book has been added to his shelf he will be redirected back to the page on which the requested started. Although this solution works it requires the page to reload each time a book has been added to a shelf. Which makes the process unnecessary slow, while the user is unable to do anything.

5 Testing

5.1 Error handling

There are 3 moments in the application flow where error handling is needed. The first moment is during the image search (5.1.1). The second moment is during the addition book information retrieval moment on the book page (5.1.2). Finally, during the online offer retrieval moment, which also occurs on the book page (5.1.3).

5.1.1 Errors during the image search

There are two things that can go wrong in this phase. Firstly the user can have uploaded an image that does not contain a book. Secondly the Google Books API might not be able to find the book in the image (Figure 5.1.1.1).

```
export async function searchByImage(src: string): Promise<IBookData> {  
  
    const visionResponse = await GoogleVisionService.analyseSingleImage(src);  
    if (!GoogleVisionService.imageContainsABook(visionResponse)) {  
        throw new Error( message: "Image does not contain a book.");  
    }  
  
    const queryText = visionResponse.fullTextAnnotation.text;  
    const googleBooksResponse = await GoogleBooksService.findBookByQuery(queryText);  
    if (googleBooksResponse.length === 0) {  
        throw new Error( message: `Could not find a book based on the text: ${queryText}`);  
    }  
  
    const googleBookData = googleBooksResponse[0];  
  
    return await composeData(googleBookData);  
}
```

Figure 5.1.1.1 The error check in the *BookSearchServices*

When one of these scenarios occurs an error will be thrown which will be caught by the controller (Figure 5.1.1.1). The controller will return a 400 status code with an error message. This message is used by the client side to indicate to the user what went wrong.

```
BookSearchController.post( path: "/image", upload.single( fieldName: "file"), async (req: Request, res: Response) => {
  const filePath = req.file.path;
  try {
    const result = await BookSearchService.searchByImage(filePath);
    res.json(result);
  } catch (e) {
    res.status( code: 400).json( body: {
      message: (e as Error).message,
    });
  }
  fs.unlink(filePath, callback: () => {});
});
```

Figure 5.1.1.2 The error handling in the *BookSearchController*

The client side will inform the user what went wrong using a model that displays the error message (Figure 5.1.1.3).

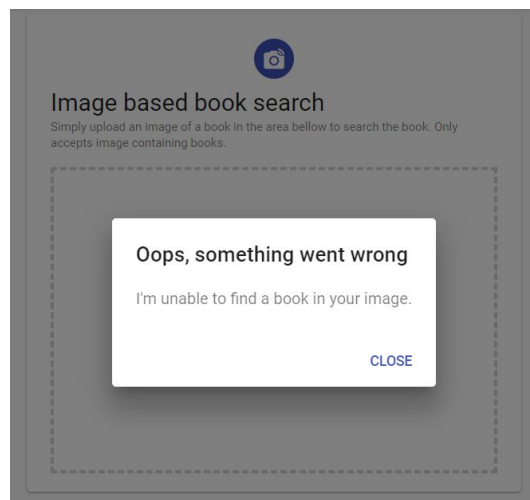


Figure 5.1.1.3 The error handling display in the UI

If for some reason the Google vision API is unavailable the user will be unable to perform image searches. However the user can still visit the book page he has already found based on their url, since the book page does not depend on the google vision API.

5.1.2 Goodreads cannot find the book

When we are unable to retrieve the data from the Goodreads API, for whatever reason, the UI must be able to handle it. Three parts of the book page depend on this data. So these parts must be updated when this error occurs.

Firstly, the “Have read” and “Want to read” buttons will become disabled. I choose to disable them instead of removing them because removing them would result into a major jump in the page layout which will be noticed by the user (Figure 5.1.1.3 left).

Secondly the reviews part of the page also depends on the Goodreads API. When the API is unavailable the review area will display the message “There are no reviews available for this book” (Figure 5.1.1.3 center).

Finally the similar books part of the page also depends on the Goodreads API. This data is retrieved through an async request that takes a while to load. So if this requests fails we will simply leave the similar books area empty (Figure 5.1.1.3 right).

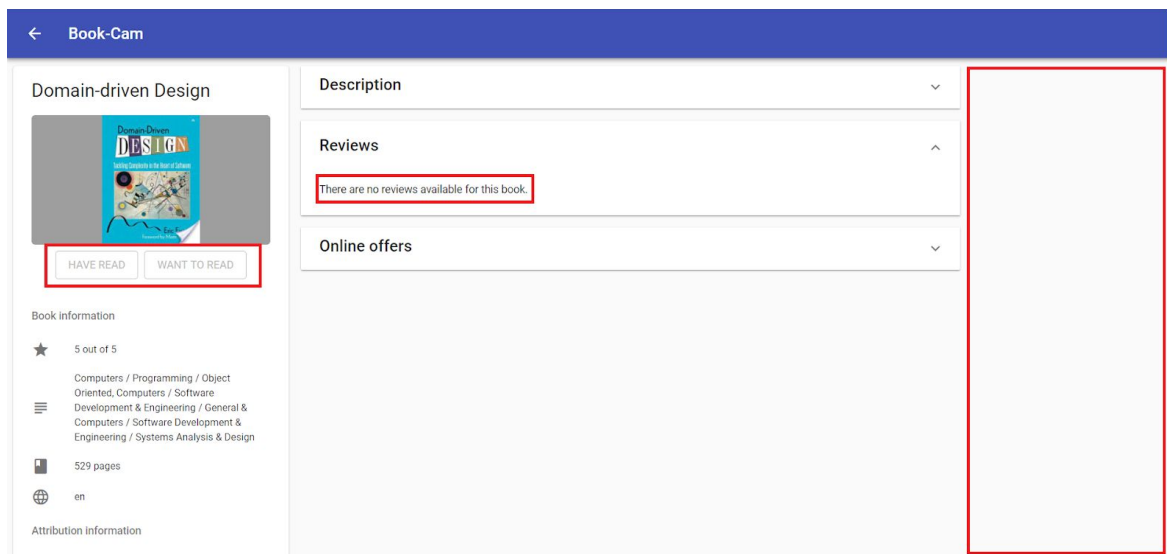


Figure 5.1.2.1 UI when there is error with the GoodReads API

5.1.3 No online offers could be found

The online offers part of the page depends on the data from the Bol.com part of the API. The UI must be able to handle it when there are no offers available or when the API is unreachable. When one of these events occurs the online offers area will display the message “There are no online offers available for this book.” (Figure 5.1.3.1).

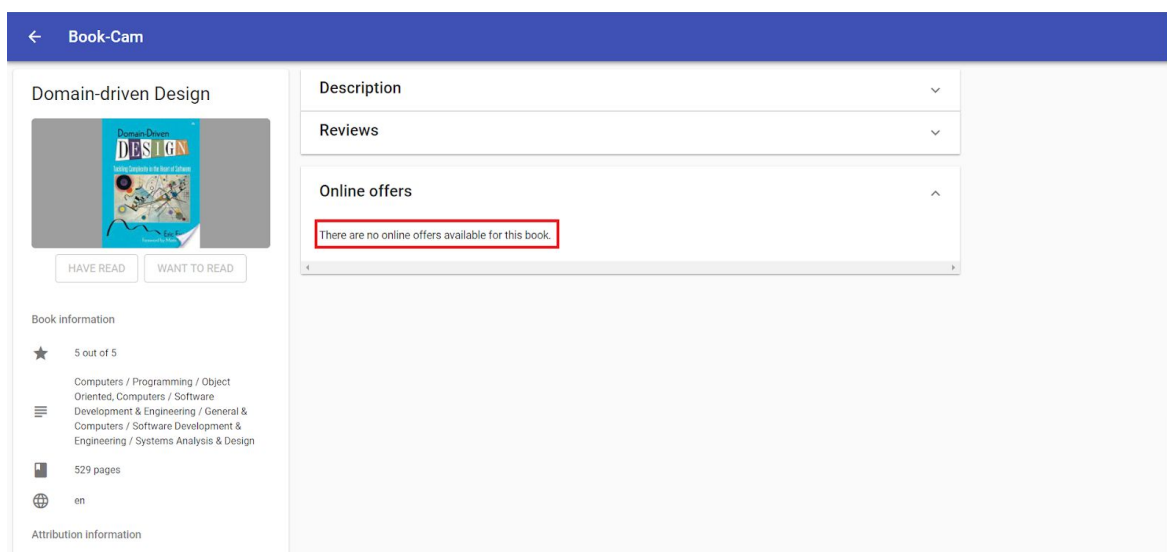


Figure 5.1.3.1 UI when there is error with the bol.com API

5.1.4 Uncaught error on the client side

It is still possible that something we have not thought about might go wrong. If this happens we don't want the user to see a frozen white screen. Therefore if there is any uncaught error (Figure 5.1.4.1) on the client side the user will be redirected to the 404 page (Figure 5.1.4.2 The 404 page).

```
public componentDidCatch(error: any, info: any): void {  
  location.href = "/404";  
}
```

Figure 5.1.4.1 The general error catch

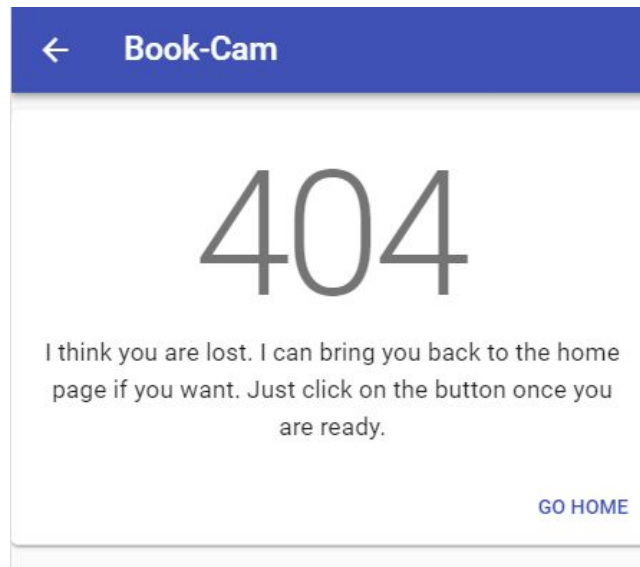


Figure 5.1.4.2 The 404 page

5.2 test cases

While testing the application manually, I checked the following functionality test case.

Task	Expected Outcome	Result	Visualization
Search for a book using an image of that book.	The users should be redirected to book page. This page should contain information about the book in the image.	Pass	Figure 2.1.1/ 2.1.2
Search for a book using an image that does not contain book.	This should show the user an error popup.	Pass	Figure 5.1.1.3
Click on the <i>Description</i> tab.	The user should now see a description of the book.	Pass	Figure 2.1.3
Click on the <i>Reviews</i> tab.	The user should now see some reviews of the book.	Pass	Figure 2.1.3
Click on the <i>Online offers</i> tab.	The user should now see an overview of online offers for this book.	Pass	Figure 2.3.1
Click on one online offers.	The user should be redirected to the webpage of the online offer.	Pass	Figure 2.3.2
Click on the <i>Want to read</i> or <i>Have read</i> button.	After login into goodreads the book should have been added to the <i>Want to read</i> or <i>Have read</i> shelf.	Pass	Figure 2.2.3
The user navigates directly to the book page	The page should load directly without the user having to provide the original image.	Pass	Figure 2.1.3
The user navigates to a none existing page.	The user should be shown a 404 page.	Pass	Figure 5.1.4.2
The users encounters uncaught error.	The user will be redirected to the 404 page.	Pass	Figure 5.1.4.2
The Goodreads API is unavailable	The shelf buttons will be disabled and the page should still load.	Pass	Figure 5.1.2.1
The Bol.com API is unavailable	The page should still load.	Pass	Figure 5.1.3.1

6 Extensions

6.1 Other users have searched

It might be interesting to keep track of the searches that users have performed by storing each successful search result. These search results could be used to provide other users with a list of suggestions or most commonly searched books. This list could then be displayed on the home page next to the actual search form.

This data can also be used to provide each user with an overview of previous searches and provide the user with suggestions based on his search history.

6.2 Additional online vendors

Currently the online offers overview only provides offers from Bol.com and its subsidiaries. This limits the amount of information about online offers the user will be provided with. In the future additional information could be added from other online vendors such as Amazon in order to improve the online offers overview.

The Amazon API was not used in this project because I was unable to retrieve an API key for this project.

6.3 Allow text based searches

There might be users that find the information overview of this project very useful. However these users are unable to get this information without having an image of the book they're interested in. For these users it will be useful to allow text based searches. Which is a feature that is very doable but is not yet implemented.

Bibliography

API. (n.d.). Retrieved August 23, 2018, from <https://www.goodreads.com/api>

bol.com | de winkel van ons allemaal. (n.d.). Retrieved August 23, 2018, from <https://www.bol.com>

Google Books APIs | Google Developers. (n.d.). Retrieved August 23, 2018, from <https://developers.google.com/books/>

Home - Bol.com Developer Center. (n.d.). Retrieved August 23, 2018, from <https://developers.bol.com/>

npm: goodreads-api-node. (n.d.). Retrieved August 23, 2018, from <https://www.npmjs.com/package/goodreads-api-node>

npm: google-books-search. (n.d.). Retrieved August 23, 2018, from <https://www.npmjs.com/package/google-books-search>

npm: @google-cloud/vision. (n.d.). Retrieved August 23, 2018, from <https://www.npmjs.com/package/@google-cloud/vision>

React – A JavaScript library for building user interfaces. (n.d.). Retrieved August 23, 2018, from <https://reactjs.org/index.html>

React Router: Declarative Routing for React. (n.d.). Retrieved August 23, 2018, from <https://reacttraining.com/react-router>

TypeScript - JavaScript that scales. (n.d.). Retrieved August 23, 2018, from <https://www.typescriptlang.org/>

Vision API - Image Content Analysis | Cloud Vision API | Google Cloud. (n.d.). Retrieved August 23, 2018, from <https://cloud.google.com/vision/>

webpack. (n.d.). Retrieved August 23, 2018, from <https://webpack.js.org/>

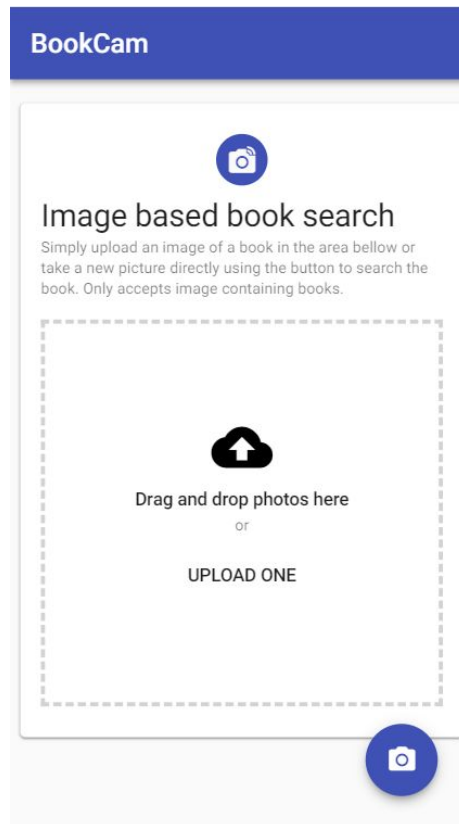
Appendix A Docker deployment

In order to deploy the application using docker run the following command in the folder container the Dockerfile:

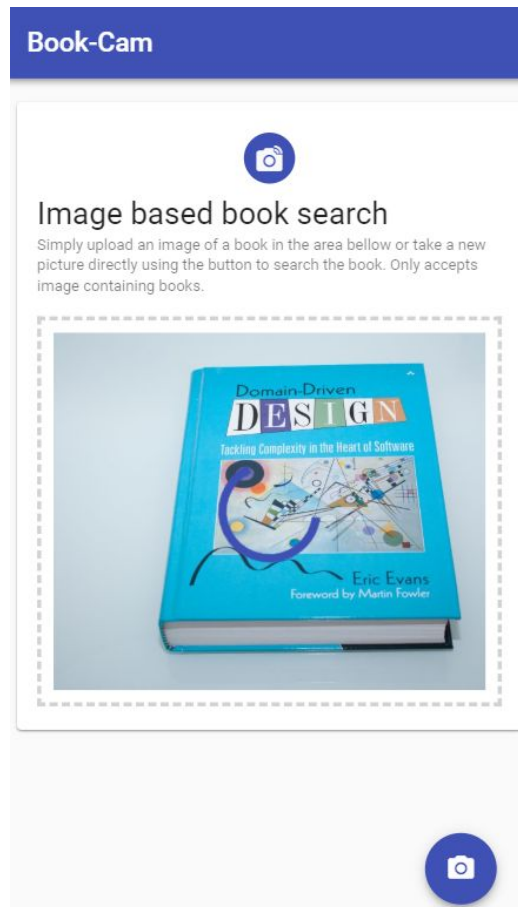
```
docker build -t <IMAGE_NAME> .  
docker run --name=<CONTAINER_NAME> --publish=80:8080 <IMAGE_NAME>
```

Appendix B User guide

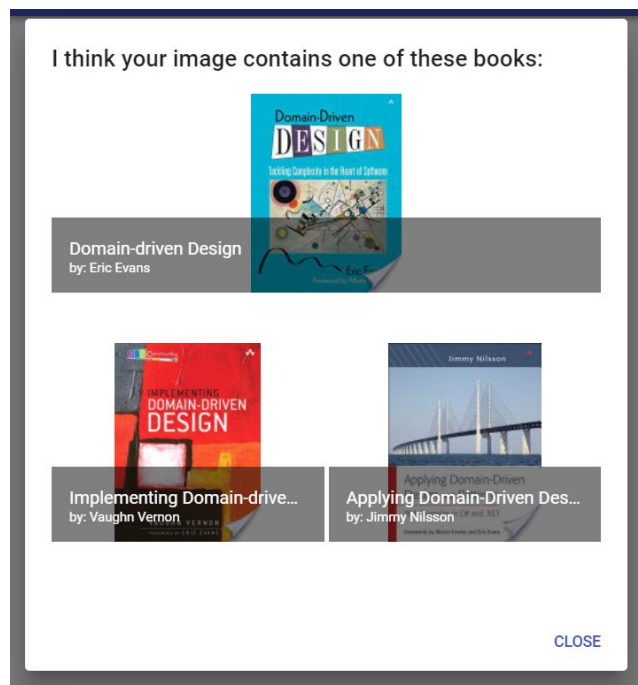
Step 1 the user navigates to: <http://book-cam.com>



Step 2 the users uploads an image by clicking the upload button or dragging and dropping an image on the dropzone



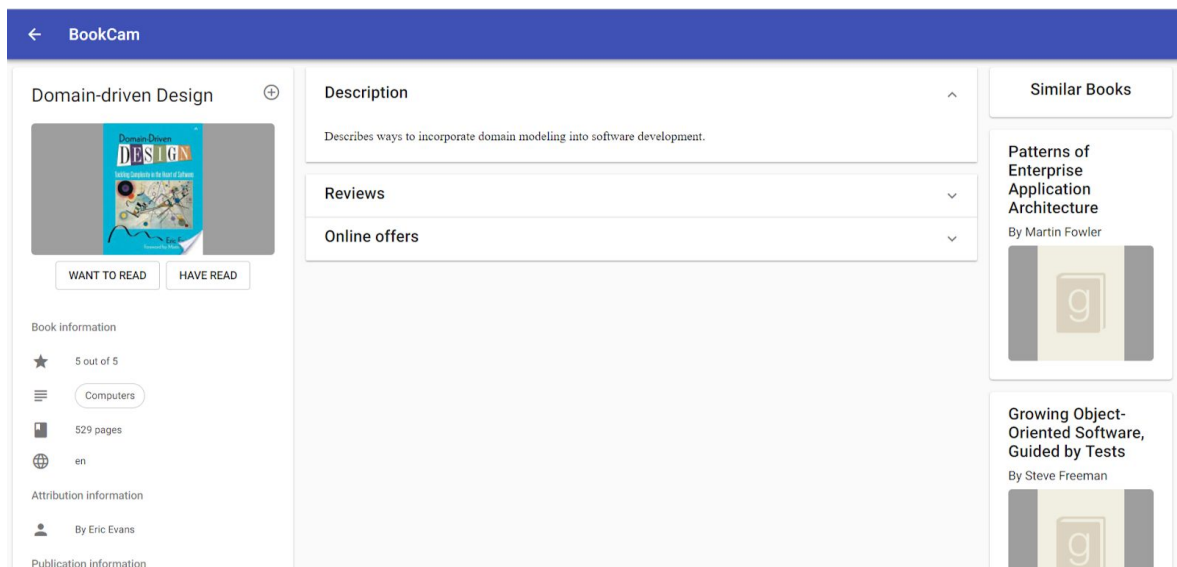
Step 3 the users selects the correct query result.



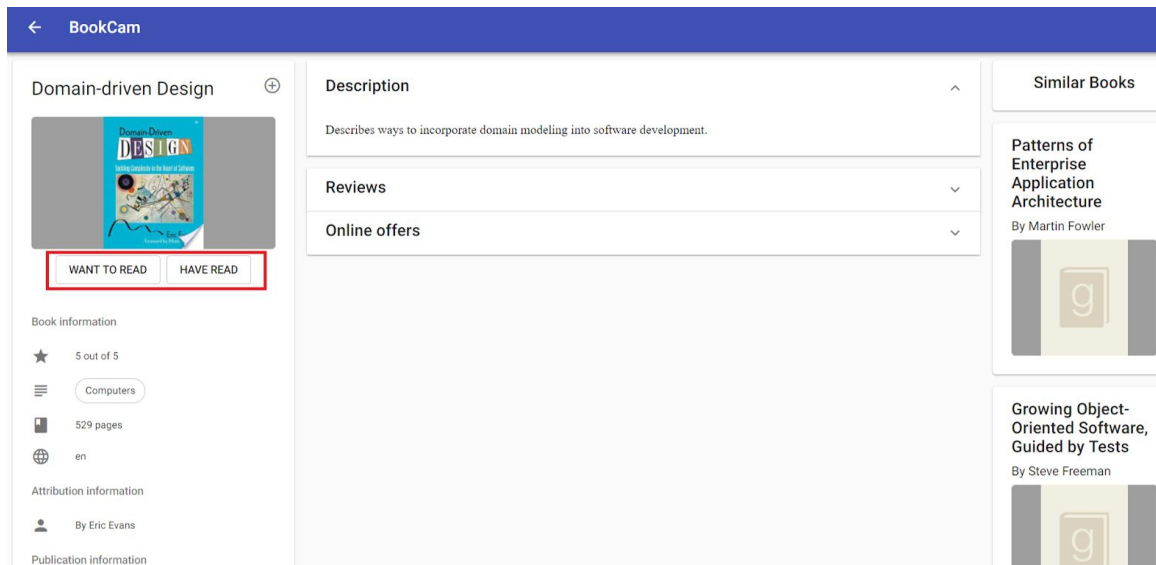
Step 4 The user finds the information he interested in on the book page

If you don't have a valid book image navigate directly to the book page using:

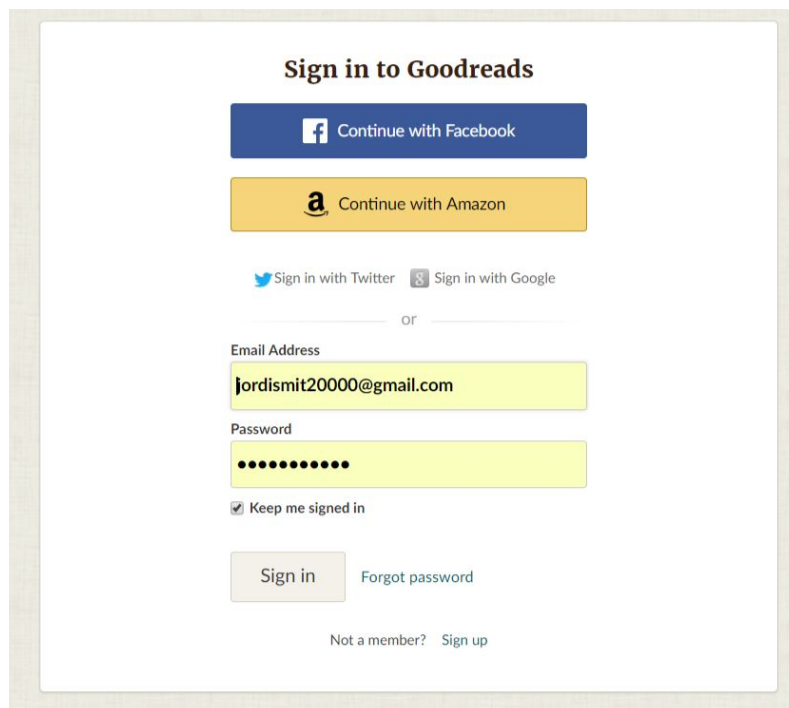
<http://book-cam.com/book/xColAAPGubgC>



Step 5 The user add the book to its Goodread shelf



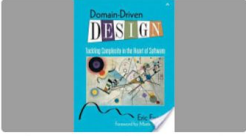
Step 6 The user logs in with Goodreads



Step 7 The book has been added to the user's shelf

← BookCam

Domain-driven Design



WANT TO READ

HAVE READ

Book Information

★ 5 out of 5

Computers / Programming / Object Oriented

Computers / Software Development & Engineering

Computers / Software Development & Engineering

📖 529 pages

🌐 en

Attribution Information

Description

"Eric Evans has written a fantastic book on how you can make the design of your software match your mental model of the problem domain you are addressing.

"His book is very compatible with XP. It is not about drawing pictures of a domain; it is about how you think of it, the language you use to talk about it, and how you organize your software to reflect your improving understanding of it. Eric thinks that learning about your problem domain is as likely to happen at the end of your project as at the beginning, and so refactoring is a big part of his technique.

"The book is a fun read. Eric has lots of interesting stories, and he has a way with words. I see this book as essential reading for software developers--it is a future classic."

--**Ralph Johnson**, author of *Design Patterns*

"If you don't think you are getting value from your investment in object-oriented programming, this book will tell you what you've forgotten to do.

"Eric Evans convincingly argues for the importance of domain modeling as the central focus of development and provides a solid framework and set of techniques for accomplishing it. This is timeless wisdom, and will hold up long after the methodologies du jour have gone out of fashion."

--**Dave Collins**, author of *Designing Object-Oriented User Interfaces*

"Eric weaves real-world experience modeling--and building--business applications into a practical, useful book. Written from the perspective of a trusted practitioner, Eric's descriptions of ubiquitous language, the benefits of sharing models with users, object life-cycle management, logical and physical application structuring, and the process and results of deep refactoring are major contributions to our field."


--**Luke Hohmann**, author of *Beyond Software Architecture*

"This book belongs on the shelf."

Similar Books


Patterns of Enterprise Application Architecture

By Martin Fowler



Growing Object-Oriented Software, Guided by Tests

By Steve Freeman



Has been added to your want to read shelf.

23