

INF-2900: SOFTWARE ENGINEERING

SCHEDGE - FIND THE TIME THAT WORKS FOR *ALL* OF YOU

ZiG Inc

May 25, 2021

Abstract

Planning events with your friends can be challenging, especially when your schedules seemingly always clash. This report describes a website which aims at finding the time which works the best for the most people. We introduce an algorithm based on a PhD thesis by Keith Ansel Marzullo (1984) to find the optimal time to host the event. A thorough description of the software development process using agile is also presented, as well as reflections around the process and the result.

Contents

1	Introduction	1
1.1	What problems are we solving?	1
1.2	Product vision	1
1.3	How will it impact the user?	1
2	Background	1
2.1	General Data Protection Regulation	1
2.2	Our competitors	3
2.2.1	Outlook	3
2.2.2	Facebook	3
2.2.3	Doodle	3
2.3	Marzullo's algorithm	4
3	Methodology	4
3.1	What types of data do we collect	4
3.2	How we analyze it	4
3.3	Agile Software Development	4
3.3.1	Scrum	4
3.3.2	Extreme Programming	4
3.3.3	Test-Driven development	5
4	Design and architecture	5
4.1	User Interface and User Flow	5
4.2	Database design	5
5	Implementation	5
5.1	Dependencies	5
5.1.1	django-notifications	5
5.1.2	Bootstrap	6
5.1.3	AJAX	6
5.2	Django	6
5.3	Events	6
5.4	Users	6
5.5	Time Slots	6
5.6	Potential Time Slots	6
5.7	Notifications	7
5.8	Friends	7
5.9	Riise Hofsføy Algorithm	7
5.9.1	Algorithm	8
5.10	Splitting a time slot	8
5.11	Time slot validation	8
5.12	General Data Protection Regulation	9
5.12.1	Active measures	9
5.12.2	Consideration	10
6	Software Development Process	10
6.1	Sprints	10
6.2	How we worked during the project	10
6.2.1	DevOps and Code Management	10
6.3	Test-Driven Development	11
6.4	Continuous Integration	11
6.5	Development of the user interface	11
6.5.1	Functional stage	11
6.5.2	User-friendliness stage	11
6.5.3	Refining stage	12
7	Experiments and analysis	12
7.1	Riise-Hofsføy complexity analysis	12
7.2	Test and coverage - charts	12
7.3	Single-user performance tests	13

8	Discussion	13
8.1	Justifying data collection	13
8.2	Practicality of the Riise-Hofsøy algorithm	13
8.3	Benefits/disadvantages of the agile methods used	13
8.3.1	Test-driven development	13
8.3.2	Pair-programming	13
8.3.3	Incremental planning	14
8.3.4	Collective ownership	14
8.3.5	Continuous integration	14
8.4	What we have learned	14
8.4.1	Proper CI/CD:	14
8.4.2	Proper implementation of tests	14
8.4.3	Development tools	14
8.5	Alternative methods that could have been used	14
8.5.1	Sub-sprints within sprints	14
8.5.2	Waterfall development	15
8.5.3	Increased use of user tests	15
8.6	Future plans	15
8.6.1	Mail server	15
8.6.2	Chat function	15
8.6.3	Picture feed	15
8.6.4	Integrated calendar	15
8.6.5	Integrated map	15
8.6.6	Better user profiles/friends system	15
8.6.7	Business facing	15
8.6.8	Extend the duration of an event	16
8.6.9	Automated data cleaning	16
8.7	Results	16
9	Conclusion	16
10	Acknowledgements	16

1 Introduction

This report will explain the agile development process, implementation and use of a web-based product for scheduling events based on finding available time slots for attendees. It will go over the technical aspects of the project, such as the database setup and its relations, important algorithms used for time management and security restrictions etc. As well as interface problems, such as the user-friendliness of the interface. The methods used for an efficient agile development process will also be discussed in greater detail.

1.1 What problems are we solving?

A problem that often arises when trying to plan an event of any capacity, both for private or organisational use, is the problem of scheduling the event for a specific time slot that fits for all/most of the people attending. When the amount of attendees for the event increases, the amount of time slots that will work for most attendees also decreases. This leads to difficulty scheduling and deciding on a particular time that fits best for all attendees. Scheduling conflicts are to a greater extent simplified by our web-based scheduling interface that allows for every invited attendee to add their preferred time slots to a non-determined event. Our customized algorithms will then calculate the overlapping time slots so that the host of the event can pick a time that fits best for all attendees within the overlapping time slots.

1.2 Product vision

For both private users and organizations **who** want to schedule events. **Schedge** is a web-based service **that** simplifies the planning and scheduling of events **unlike** other services or package software products, **our product** provides a more inclusive planning process by taking into consideration when the event will fit best for most attendees.

1.3 How will it impact the user?

This service will emphasize the needs/wants of all its users, instead of only the host of the event, by making everyone's suggestions highlighted and easy to choose from. The customized algorithms will effectively find the most fitting time-slots, listing them in order of best fitting. This makes for a more inclusive and simpler way of deciding a time and date of a specific event with all attendees in mind.

2 Background

2.1 General Data Protection Regulation

General Data Protection Regulation, or GDPR, is a set of rules for how a company can collect and process data from a user, and how this should be done. It also states what rights the user has to their data, and how an organization or company should comply with these (European Commission 2016).

- Organizational obligations:

- Lawfulness, fairness, and transparency:

This means that the reasoning for a company to process data from a user needs to fit into at least one of these six categories: Consent, Contract, Legal Obligation, Vital interests, Public task or Legitimate Interests.

For the processing to be lawful, there needs to be considered a requirement to process the data. This means considerations of requirements needs to happen before the collection and processing of data occurs.

Another part of this is that a company cannot expand their use of the data, without the user's explicit consent. If data is collected and processed for the use of the service, then the same data cannot be used in marketing or other applications.

- Purpose Limitation:

A clear description of why specific data points are collected, and what they will be processed and used for, needs to be present and available to the user. Since a user can opt out of this, consent has to be given before data collection begins.

- Data Minimization:

A big part of GDPR is that companies should minimize the amount of data collected about a user. The data points collected needs to be deemed necessary, and cannot be collected without any thought for why it is needed. A company should be able to argue why some data point is collected, and be able to inform about the concrete reason if asked for it. For example, collecting a user address would only be valid in the case where the company/service is shipping something to the user.

Some key measures to consider when minimizing the data collection:

- * Adequate
- * Relevant
- * Limited

Although data collected should be kept at a minimum, adequate and relevant data points to complete the processing needs to be collected to avoid partial data. This would be the same as collecting more data than necessary, as there would not be enough data points to complete the processing.

- Accuracy:

GDPR tries to ensure that the data collected is correct, and not falsely stated. Depending on the data points a company collects, and in what application it will be used, they may need to verify that the data a user provides, is correct.

- Storage Limitation:

A company should not keep data that is no longer necessary. There should be in place a policy or routine to clear out unused data and inactive users. Additionally, a user should be able to review or delete the data collected about them, upon request.

Smaller organizations don't need a documented retention policy, but the requirement to review data and delete any data that is no longer needed still applies.

- Integrity and Confidentiality (Security):

To uphold the integrity and confidentiality of the data, some key requirements have to be followed. One is that only those authorized to do so can access, alter, disclose or delete the held personal data and then only to complete the tasks which have been identified and authorized. The data meet the requirements for processing in that it is both accurate and complete. And all data is both accessible and usable with systems in place to recover it should it become lost, altered or destroyed. The data should also be protected with security measures proportional to the risk of a data breach.

If a data breach occurs, where personal data is involved and people put at risk, the organization has 72 hours to report the breach to that country's information commissioner. Even if not all the information about the breach is available, it should still be reported.

- Accountability:

GDPR requires that a company or organization can demonstrate that they comply with the GDPR rules and regulations.

A *Data controller* has the ultimate responsibility for making sure the organization or company is GDPR compliant, while a *Data protection officer* will be able to guide the employees to ensure compliance. Smaller organizations can meet the accountability requirement by ensuring that everyone understands the importance of, and need for, data protection and the impact it can have on users. This needs to be combined with policies and procedures to handle personal data, and records of what is processed, and why.

- User rights:

- Right to be Informed:

A user has the right to be informed about what data is being collected about them and how it will be processed. The user also needs to be informed about how to get the information that is already collected. In the case where data is shared with a third party, the user should be informed no later than when the sharing takes place.

- Right of Access:

An individual has the right to request access to all data about them. This should include information

about the purpose of the processing and for how long the data will be retained. As the request is not required to go through a specific person in the organization or company, all employees need to know the importance of passing on the request to the right employee. This is to ensure the user gets access to what they have the right to know.

- Right to Rectification:

A user has the right to request rectification of information. In that case, the organization is required to consider any argument and evidence provided by the user. The organization have to take reasonable steps to either confirm that the data is already correct or, if necessary, rectify it.

- Right to Erasure:

Every user has the right to have data about them deleted. This can be the choice to withdraw their consent to data collection, or if the data is no longer needed for its intended purpose.

Should a request to have a user's data deleted, the organization has one month to complete the request. The request is not required to come through the exactly "right channels" to be valid and can be made by any means.

- Right to Data Portability:

This gives the user the right to take what data have been collected about them, from one service to another. This goes for all the information an organization has about a user, not only their profile, but search history, and other collected and processed data.

- Right to Object:

A user has the right to object to the collection and processing of their data. This requires the organization to have mechanisms in place to ensure that they can comply with these requests.

2.2 Our competitors

2.2.1 Outlook

Outlook is Microsoft's email and calendar app (Microsoft 2021). With Outlook you can create events and invite other people using their email or Microsoft account. Unlike our product, the host of the event will set the date and time of the event when it is created. There is no way to get input from your guests, so you might pick a time that does not work for several of them. This way of scheduling events works well in a business setting, where the boss knows the schedule of their employees. If the entire team uses Microsoft's calendar app, the boss will see potential conflicts when creating the event.

Our product is more focused on private events. People do not have their calendars constantly up-to-date on everything they do, so Microsoft's approach to finding a fitting time does not work.

2.2.2 Facebook

Facebook events, on the other hand, are focused more on big public events. Similar to Outlook, the host initially picks a time and date and Facebook users press, "going", "interested" or "not going" based solely on that time. When scheduling big events, optimizing the time based on how many of your guests can attend can be impractical. Instead, one usually has to pick a date very early on in planning the event to organize all the people involved. It is, therefore, impractical to use a system like the one we use in our product. Using Facebook events is not ideal for planning a small private gathering.

Additionally, Facebook has been ridden with privacy controversies like the Cambridge Analytica scandal where a voter-profiling company had gained access to 50 million Facebook users (Meyer 2018). Using Facebook to schedule your events could compromise your privacy. Most other apps require information about age, gender and so on, to maximize their ad revenue. Schedge takes privacy very seriously and does not store unnecessary data about users.

2.2.3 Doodle

Doodle is an event scheduler based in Switzerland, which, similarly to Schedge, lets the host gather information from the guests before setting the final time (Doodle 2021). When the host creates an event, they define some options for the guests. The guests of the event vote on the time that works the best for them. Based on the votes, the host can decide the final time. This differs from Schedges design as the host only sets the time limits on when the event will take place, not individual options. Doodle's approach probably works better if there is a

limited number of possible times, like if you want to see a movie at the movie theatre some weekend. However, if there are looser limits on when the event can take place, we think Schedge's approach is more practical, as the host does not know the schedule of the guests a priori. The host can gather other possibilities from the guests, and pick between them.

2.3 Marzullo's algorithm

An algorithm created by Keith Marzullo in 1984 as part of his PhD "Maintaining the time in a distributed system". The algorithm is used to find the greatest overlap of multiple intervals. To do so, every interval is split into two tuples. The first tuple indicating that the interval is starting and at what time, and the second tuple indicating that the interval is ending and at what time. After splitting each time interval into tuples, they are sorted based on their time value, which can be done in $O(n \log n)$. The tuples are then traversed, increment a counter for every tuple where an interval starts, and decrements when an interval ends. Every time the counter reaches a new top value, it is indicated that the best interval starts at the current tuple, and stops at the next. Since the tuples are traversed only once, the complexity is $O(n)$, meaning that the most computationally expensive part is sorting the tuples. (Marzullo 1984)

3 Methodology

3.1 What types of data do we collect

Schedge collects the username, name and email of its users. The username is used to uniquely identify a user, while the email is used in the case of a forgotten password. For events, the location, dates and times and a potential image are stored in the database. Schedge collects the times that the host, or any of the participants, give when they create a time slot of available times to attend. There is also stored information about who is friends with whom.

3.2 How we analyze it

The username works as a publicly known identification on the service. Allowing users to find each other, add friends and invite each other to events. For the back-end, a user-id, which is invisible to the front-end, is used to find a specific user instead of the username.

Time slots are analyzed to calculate overlapping times where multiple people are available to attend. This is done using equation (2), which returns a set of overlapping time slots where the duration is longer than the event they are intended for.

3.3 Agile Software Development

Agile software development is a development process for software products that includes a set of methods and practices (Sommerville 2019, p. 20). Agile makes development teams more equipped to respond to change, develop and deliver software incrementally, involve the customer in the development process and maintain simplicity in the code. Some frameworks within Agile development are Scrum, Extreme Programming and Test-driven development which is a sub-category of Extreme Programming. In this development process, principles from all these frameworks have been used.

3.3.1 Scrum

Scrum as a framework gives a more structured development process, as it includes some overhead of the project such as the Product Owner and the Scrum Master of the project. The Product Owner takes responsibility for the product and ensures that everyone is focused during the development phases. The Scrum Master acts like a "coach" in the sense that he will guide the team in the effective use of the scrum methods. Some other Scrum methods are the use of product backlogs, sprints, velocity and sprint backlogs. Backlogs contain all the features/tasks to be done for the project. A sprint denotes a short period where the self-organizing team will work on a product increment. The velocity is the estimate of how much work can be done in a single sprint and the sprint backlog contains the features/tasks to be implemented for the specific sprint (ibid., p. 27-28).

3.3.2 Extreme Programming

The goal of Extreme Programming is to take the incremental programming standard of Agile development to the "extreme" to to get a more efficient development process (ibid., p. 24). This is done by introducing

more demanding/extreme methods for development such as test-driven development, incremental planning, continuous integration, refactoring of code, small releases, pair programming, simple design, collective ownership, on-site customer and collective ownership of the product.

3.3.3 Test-Driven development

Test-Driven development is a method that encourages the implementation of tests for a given feature before implementing the feature. It relies on automated tests, so that every time a feature is to be added to the product increment, a test needs to be developed and passed before the feature can be approved (Sommerville 2019, p. 280-284).

4 Design and architecture

4.1 User Interface and User Flow

The flow of our website is modelled in Figure 1. When a new user first opens the website they are directed to the landing page. From here, a user who already has a profile will press sign in, while a new user will press sign up. One can also go to the *about us* page to learn more about who the people behind the website are. After logging in, the user is taken to their home page. Here, the user will see their upcoming events, pending invitations, the event they host, and so on.

From the home page, a user can either visit an existing event or create a new one. After creating an event, you are taken to the new event. From an event page, you can edit the event, which will take you back to the event after submitting the changes. From the event page, you can also go back to your home page. Signing out or deleting your account takes you back to the landing page.

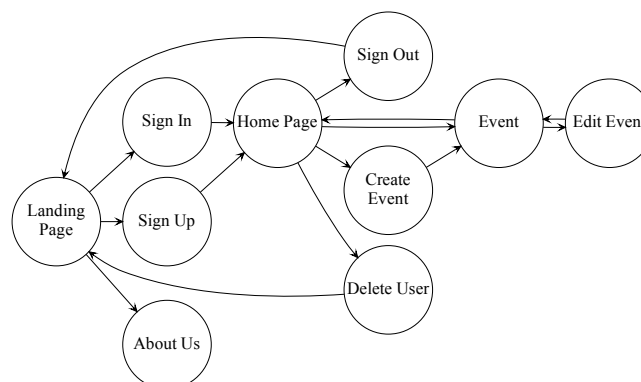


Figure 1: Flow chart

4.2 Database design

Storage of data is done in an SQLite3 database. The entity relationships are visualized in Figure 5. Both the user and the notification tables are the default Django native models. The Django user model is non-trivial to edit, so to add functionality such as friends to this model, a new model with a one-to-one relationship is used. This new model works as an observer to the user model and its corresponding entry updates itself when a user entry is updated.

5 Implementation

5.1 Dependencies

5.1.1 django-notifications

django-notifications is a Django app for adding GitHub like notifications to your site (Leonel et al. 2020). We use this app to implement our notifications. While the app had most of the functionality we needed, we have written our own callback methods for displaying the notifications, to make the notifications look nicer.

The app has a feature where it refreshes the notifications every 15 seconds. It sends a new request to the server to ask for up-to-date notifications. Then, our callback method is called to update the notifications on the screen.

5.1.2 Bootstrap

The front-end was developed with the use of the CSS framework Bootstrap. Using Bootstraps own templates for different elements simplified the process.

5.1.3 AJAX

AJAX is a technology to create asynchronous requests on the client-side. These request can be used to update the contents on a web page without having to refresh the page (Mozilla 2021).

5.2 Django

The main framework used in this project is Django. Django delivers abstractions such as easier database queries, authentication of users and URL management. The abstraction from the database queries removes the necessity for interacting directly with the underlying SQLite3 database. Django has built-in URL mapping which simplifies maintaining endpoints of the website. Because of the Django authentication functionality, the need for implementing security measures such as CSRF token and session keys is relieved.

5.3 Events

In the event model, information about the event like its title, location, duration, image and description can be found. The event also has a time interval, which specifies the earliest and the latest time at which the event can occur. Similarly, it has a date interval which is the interval of dates between which the event may take place. These intervals specify when the event can happen. The actual time and date of the event will be specified by the host after gathering information from the guests.

The event also has access to its participants through a many-to-many relationship with the user model, as an event can have many participants and a user can attend many events. When a user joins or leaves the event, this table is changed.

Moreover, the event has a status. When an event is initially created, its status is *unresolved*. An unresolved event does not have a date and time where the event will finally take place. After the host has gathered available times from the guests and determined a time, the event's status is changed to *chosen*. Finally, the event will be *finished*, when the end date of the event is in the past. Whenever a user accesses the event, we check if it is in the past, and if it is, we change its status.

5.4 Users

Users are defined similarly to Django's standard user model. Both the models use username, first name, last name, email and a password for sign up authentication. What differentiates the two models, is that the user model used in this project, in reality, is a regular model with a one-to-one relation to the Django user model. This allows the model to keep the authentication features, while also adding other fields to the model. In this case, a field to keep relations with a user's friends.

5.5 Time Slots

A time slot is a time interval that a user can attend an event. A time slot has a start, an end, and a date, which defines a time interval where the user says to the host that they are available.

A time slot can be *rollover* if it spans more than one day, i.e. it goes over midnight. If a user adds a time slot where the end is "earlier" than the start, we assume it is rollover, for example, a time slot between 10:00 PM and 3:00. As we will see, rollover time slots complicate figuring out if a time slot is within the time set by the event.

Time slots can also be merged. If a user first adds a time slot between 1:00 PM and 3:00 PM and then adds another one between 2:00 PM and 4:00 PM, we can merge them into a single one between 1:00 PM and 4:00 PM.

5.6 Potential Time Slots

If two or more different users add overlapping time slots, a *potential time slot* is created. The model of a potential time slot says for whom the time slot is available. The host will pick between these potential time

slots to find the final time. They will be able to see who the time slot is available for and for how many, to make the most informed decision. How these overlaps are calculated is explained in Section 5.9.

5.7 Notifications

Notifications are used to give users updates related to their account. There are notifications for incoming friend requests and event invitations, accepted invitations to events you host, when you are removed from an event, and so on. A notification object has some attributes:

- Actor: The user who made the notification to be sent
- Recipient: The user or users to whom the notification is sent
- Verb: A description of the action
- Action Object: An object linked to the notification, for example, an invitation
- Target: The object the action object acts upon, like what event you are invited to

Say, Alice invites Bob to go bowling; the actor is Alice and the recipient is Bob. The verb is a general description to display the notification correctly, so this is “event invite”. The action object is the event invitation and the target is the event object itself.

The purpose of the action object is to remove the notification if the event was handled in some other way, for example, if a user accepted a friend request through the friends tab on the home page. In this case, we want to remove the notification even though it was never pressed. By having a unique action object id for all notifications, we can query the database for the correct notification and remove it. The target is useful for making the notification act on a particular object. For example, accepting an event invitation through a notification should add the user to that event.

The action object is also used with *silent notifications*. A silent notification happens when an invitation is revoked. For example, if Alice sends a friend request to Bob, but quickly changes her mind and undoes the invitation. To try to avoid an awkward situation between Alice and Bob, we remove the notification that was sent to Bob without notifying him. It is, of course, possible that he saw the notification before Alice revoked the invitation.

5.8 Friends

To add a social aspect to our site, users can become friends with one another. Friends are implemented through many-to-many relationships. When a user wants to invite someone to an event, they can either write in a username or pick a friend in a drop-down menu. We assume users are more likely to add their friends to an event than anyone else. By adding this feature, we eliminate the need to memorize the usernames of all of your friends.

5.9 Riise Hofsøy Algorithm

Marzullo’s algorithm finds the optimal overlap from a set of partially overlapping intervals (Marzullo 1984). This algorithm has been modified to return every permutation of overlapping intervals. All overlaps, Ω , can be found using the equation

$$\Omega = \left\{ \bigcap p \mid p \in \mathcal{P}(A) \wedge |p| > 1 \right\} \quad (1)$$

where A is the set of all user time slots, and $\mathcal{P}(\cdot)$ denotes the power set. However, the duration of the intervals must also be larger than a predefined duration T_0 and inside a boundary interval T . These intervals will all be potential time slots for the event and is defined by equation (2).

$$\{(a, b) \in \Omega \mid \mathcal{D}(a, b) \geq T_0 \wedge (a, b) \subseteq T\} \quad (2)$$

In equation (2), (a, b) refers to some interval with start a and end b , $\mathcal{D}(\cdot)$ is the duration of some interval defined as $\mathcal{D}(a, b) = b - a$. Figure 2 visualizes an example use case of the Riise-Hofsøy algorithm. The letters $\alpha - \varepsilon$ represent user time slots, and p_i represents the expected output of the algorithm. Thus, $\Omega = \{p_1, p_2, p_3\}$, but if $p_3 < T_0$, then only p_1 and p_2 are considered valid potential time slots.

5.9.1 Algorithm

The algorithm explained in the previous sub sections are outlined in Algorithm 1.

Algorithm 1 Riise-Hofsøy

```

1: Set up list of tuples
2: Sort tuples by date (from earliest to latest)
3: Set minimum count to desired value for creating an overlap and initialize empty list to keep track of intervals
   started, but not finished
4: for each tuple in list of tuples in ascending order do
5:   if Start of interval/step up then
6:     Add the interval to the list of started intervals
7:     while End of interval is not found do
8:       if tuple is an interval ending, the length of the interval is long enough and the number of active
         intervals is equal or greater than the minimum number specified then
9:         Establish the valid overlap
10:      end if
11:    end while
12:  else ▷ End of interval/step down
13:    Remove the interval from the list of started intervals
14:  end if
15: end for
  
```

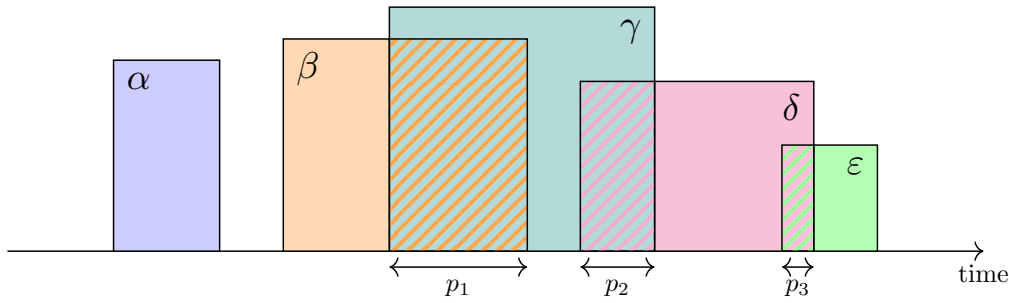


Figure 2: Riise-Hofsøy finds all overlapping time intervals

5.10 Splitting a time slot

Given some potential time slot, p , for an event, which starts at a time, s , and ends at e , we want to split it up into shorter intervals from which the host will pick. If the event lasts for some duration d , the times that the hosts can choose between is given by the following equation:

$$\{t \text{ s.t. } k | t_{min} \wedge s \leq t \leq e - d\} \quad (3)$$

where k is some interval between the times, currently 15 minutes. In other words, we find all time intervals in p which are k minutes apart and of the same duration as the event and within its start and end time. As $p \subseteq e$, we know that all t will be within the start and end times of e . The condition k divides t_{min} means, given $k = 15$, that the minute part of all the times is either 0, 15, 30 or 45. This is done to limit the number of options for the host to a more manageable number. We assume the exact starting point of most events does not need to be more accurate than this.

5.11 Time slot validation

Validating that a time slot is valid is not trivial. Because event's and time slots may span either one or two days, checking overlaps depends on both the time slot and the event the time slot is for. The simplest cases are displayed in Figure 3, where the event is contained within one day. A time slot, (t_s, t_e) is legal if it is a subset of the event's allowed time, (e_s, e_e) , like in Figure 3b. If there is any overlap with the red area, the time slot is illegal, like in 3c. Formally, a time slot is legal if it satisfies the following condition:

$$t_s \geq e_s \wedge t_e \leq e_e \quad (4)$$

When an event spans two days, i.e. $e_e < e_s$, things get more complicated. Consider the example in Figure 4b; as $t_s > e_s$, it does not satisfy (4), even though it does not overlap with any red area. To fix this issue, we introduce a dummy date, say January 1st, where all of our times initially lie. In this example, the time slot spans two days, so we add 24 hours to t_s , t_e and e_e as these are after midnight. Since t_s is on January 2nd, $t_s > e_s$ is now true. We consider e_e to be on January 2nd if the event spans two days. We add a day to t_s if $t_s < e_e$ and to t_e if $t_e < t_s$. More simply put, if the event spans two days, all times between midnight and the end of the event will be put on January 2nd.

After placing the times on the correct date, equation (4) will correctly validate all time slots regardless of rollover. The time slot may also be illegal if it is too short. A time slot must be longer than the duration of its event.

To complicate things further, we also have *all-day* events. If the start and end time is equal, an event is considered an all-day event. In an all-day event, all time slots are legal as long as they are longer than the duration of the event.

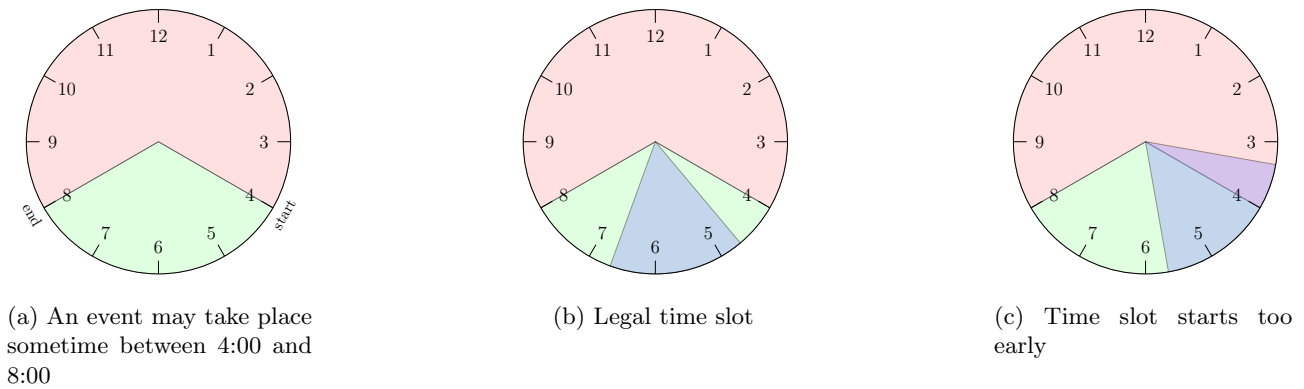


Figure 3: Proposed time slots

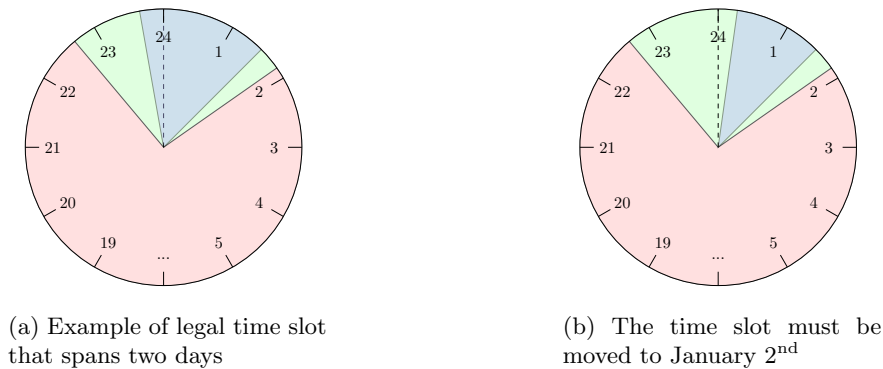


Figure 4: Legal rollover time slots

5.12 General Data Protection Regulation

5.12.1 Active measures

As users sign up, they are encouraged to read our “Terms and conditions” page. Here the terms and conditions of using Schedge, as well as information about how to contact the ZiG Inc. team, and their right to delete their account, is stated. The contact information can be used to request any data Schedge has collected about a user, which is compliant with an individual “right to be informed” and the “right of access”. Since the terms and conditions state that by using the service they accept the terms, they need to explicitly check a checkbox to accept this, before the option to complete the account sign-up is available.

Once an account is up and running, they can “Schedge” as intended. In the event where a user no longer wishes

to have an account with Schedge, they always have the option to delete their account. This is done through a button on the dashboard, with a pop-up to confirm the deletion. Upon confirmation, the logged-in account will be permanently deleted from Schedge's servers and systems, and will not be retrievable.

To prevent unauthorized access to events or other information that a user has access to on their profile, the users need to be logged in to their account to get access. This means that if they are not logged in, any events, friends, invitations or anything else they might have on their profile, is not accessible. Only the things that are accessible to everyone who visits Schedge, like the landing page, login etc. is visible to the user. When this is combined with the security measure that keeps a user out of an event. Preventing an unauthorized user from seeing the details or adding a preferred time to attend, unless they have accepted to join the event. Which, in turn, would authorize them to both see and interact with it. This makes the app more compliant with the "Integrity and confidentiality" part of GDPR.

As a way to comply with "Accountability" within the GDPR, Alexander Torkelsen is the Data Controller at ZiG Inc. and Schedge. There has also been a meeting where everyone in the team has been informed about the importance of GDPR, and what to do if a request from a user to see their data comes in. As anyone in the team can get this request, they will have to pass the request on to the Data controller which will then handle it.

5.12.2 Consideration

Only data that is strictly necessary to deliver the service is collected from the user. The obligation to be lawful, fair, transparent is upheld. But since enough data to deliver the service is collected, and no partial data set is held, this also means Schedge try to follow the obligation of "Data Minimization".

Because the users need to look at and understand the Terms and conditions, as well as agree to comply, before signing up. "Purpose of limitation" is considered with this implementation of Schedge. Since the service is still in its early stages with only a manageable amount of users, we comply with "storage limitations", by routinely and manually cleaning out unused data and removing users who have been inactive for a set amount of months. As the user base increases, a need to automate this task would become more pressing, but this is mentioned under future work.

6 Software Development Process

6.1 Sprints

Our project was developed in three sprints lasting around 4-6 weeks each. At the start of every sprint, a sprint backlog was created and the features to be done next in our project was added. The priority of the features and their velocity was taken into consideration when deciding which feature should be implemented next. When a general idea of the features to be implemented was added to the list, the team would discuss together and refine exactly how we wanted their functionalities and displays to work. For the rest of the sprint, these features were implemented by the team. Since the features were added for each sprint, this meant that the team had more flexibility to shuffle around and redefine the features needed in our project.

The sprint backlogs can be found in the appendix.

6.2 How we worked during the project

The project started by splitting the group into three teams of two. During the first sprint, the teams started with pair programming to ensure every team member got incorporated into the project. In the second iteration, it was decided that one team would split up such that one member worked with fixing issues and the other worked with the front-end. The two other teams worked with the features for the iteration. Most of the group worked from the same office, whilst the others worked in offices nearby, this resulted in low thresholds for quick meetings and questions. Over the course of the project, the teams naturally evolved into specializations like back-end, front-end, and full-stack.

6.2.1 DevOps and Code Management

During the project we have used the DevOps tool GitLab where we had a repository for the project. Each new feature would be described in an issue. A developer would then mark an issue as "work in progress" and create a new feature branch. When the feature of the branch were finished the branch would be placed in the continuous

integration pipeline to be merged into master. The use of issues assisted the developers in discovering and distributing features and bugs.

6.3 Test-Driven Development

Writing test has been an important part of this project. Normally tests are written before the functionality to ensure the credibility of the tests. This was mostly been done in this project as well, however, halfway into the project it was discovered that parallelizing this process increased productivity. This was done by having one half of a team write tests for the functionality that the other half is working on. This type of test-driven development demands proper communication inside the team. Before the development process begins, the team agreed upon input parameters and the output values of the functionality were to be implemented. In contrast to the traditional test-driven process, if one part of the team discovers a flaw in the parameters of the functionality, this could be changed without having to rewrite all the tests.

After the tests were written and the functionality had been implemented a test coverage report is generated to ensure that all lines of the new functionality are tested. This report doesn't provide any information about the quality of the tests written, but if some code is not run then some edge cases are not tested. The missing tests were written before the team marked the functionality as completed.

6.4 Continuous Integration

During the project, continuous integration was used to ensure the quality of the code submitted onto the master branch. The purpose of the master branch is to always have the latest working version of the product. All functionality on the master branch should be thoroughly tested and all tests should pass. When new functionality is ready to be merged into master, the compatibility of the current master branch and the functionality branch is checked. Thereafter, all the tests for the project are run to ensure the new functionality doesn't break some old functionality. If all tests complete without errors the functionality is ready to be merged into the master branch. The developer would then request a review from another developer. During the review, the code is checked for misspellings and readability, if the code passes the review, the functionality is merged into the master branch by the reviewer. This path that the new functionality takes from the developer's branch to be integrated into the product is referred to as the continuous integration pipeline.

When new code is merged into the master branch, reports of tests coverage and documentation coverage is generated. If the tests coverage is less than a predefined percentage a CI failure will occur and more tests have to be written immediately. In contrast, however, to comply with the principles of agile development, the documentation coverage is allowed to fail, and will only give a warning if the coverage percentage is lower than 90%.

6.5 Development of the user interface

One can divide the development of the user interface into three separate stages, one for each iteration; a *functional* stage, a *user-friendliness* stage and a *refining* stage.

6.5.1 Functional stage

The first stage showed a clear presence of a developer's mindset. The objective of this stage was to make sure that the back-end functionality gave the necessary information needed for the user interface. This resulted in a complicated interface where one needed a clear understanding of the code that lays beneath to be able to follow it. In the first iteration, this approach was sufficient since one wanted to make sure that the fundamentals were solid. When the basics of the functionality were laid, we moved on to the next stage.

6.5.2 User-friendliness stage

In the second stage, the main goal was to make it user-friendly, and this stage consisted of making the product as intuitive as possible. To do so, it was important to identify all of the different features and where on the website they should appear. The user stories created in the first iteration were extensively used for this.

The first step in this stage was to create low-fidelity prototypes that captured the gist of the user interface. An emphasis was on the general structure of the different pages and breaking down high-level tasks into several simple actions. The layout needed to be organized in such a way that the user does not need to spend excessive time familiarizing themselves with the interface. This involved finding an ideal flow of the different sections and

Table 1: Coverage report

Name	Statements	Miss	Cover
<code>schedge/forms.py</code>	100	0	100 %
<code>schedge/models.py</code>	105	18	83 %
<code>schedge/templatetags/nicedelta.py</code>	21	6	71 %
<code>schedge/templatetags/placeholder.py</code>	8	4	50 %
<code>schedge/utils.py</code>	9	2	78 %
<code>schedge/views.py</code>	400	21	95 %
<code>schedge/widgets.py</code>	43	9	79 %
Total	743	61	92%

making it clear what is expected of the users to get the desired output.

Another aspect in this stage was the principles one should follow when developing the user interface. It was important to figure out where to set the base for the users' skill level. When creating the interface one had to keep in mind who the product was intended for and their knowledge. One of the ways the interface accommodated just that was how the forms were displayed. When it came to creating an event the input of the time interval of day and the duration of the event was one of those features that could be confusing. The solution was to split the form into different sections where each section had one purpose and it was clear for the user what that purpose is. This approach was also taken when redesigning the "Add a time slot" functionality. The chosen interaction style for the website was a combination of Menu selection and Form Fillin, where the intention was simple navigation and easy event creation.

6.5.3 Refining stage

The final stage was refining the interface. The user interface at this stage of the process was user-friendly, but it lacked refinement and simplicity. Design choices such as which font to use, colour theme and button styles were the main content at this stage.

An important tool for the development of the user interface was Ben Schneiderman's eight golden rules of interface design (Schneiderman and Plaisant 2004, p. 74). One of the most important rules was the *Strive for consistency rule*, which was the main focus when refining the interface. The style of the different headers should be similar, as well as the buttons and forms. Another important rule was the *Offer informative feedback rule*. It was important to make the different actions of the site clear for the user by giving them the necessary explanations and feedback. An example of this is when a user submits an invalid form. The site uses AJAX requests to display an error to the user without the page being reloaded. If the site is not reloaded, and an error message is displayed, the user can easily change their input to make the form valid.

7 Experiments and analysis

7.1 Riise-Hofsøy complexity analysis

The complexity of Marzullo's algorithm is always defined by the sorting algorithm used. Marzullo achieves this by only looking for the optimal time slot, which is the time slot with the largest amount of overlaps. (Marzullo 1984) In contrast, the Riise-Hofsøy algorithm finds all possible permutations of overlaps between different users. This requires one more loop going from the current start tuple to the end tuple. The best-case scenario occurs when there are no overlapping time slots, if this is the case discovering that there are no overlaps is done in linear time. Therefore, as with Marzullo's algorithm, the best-case scenario for this algorithm depends on the sorting algorithm used and will normally be $O(N \log(N))$. The worst-case scenario occurs when the time slots are ordered as a matryoshka doll, meaning that all time slots are nested inside a larger time slot. In this scenario, the inner loop will be forced to iterate from the point of the outer loop to the end of the tuple list. The number of iterations will then be $N + (N - 1) + (N - 2) + \dots + 1$, this is a series known as the triangle numbers, and has a complexity of $O(N^2)$.

7.2 Test and coverage - charts

Table 1 shows the results of the final coverage report. This report describes what lines of code that is run by the tests. Statements refer to the total number of executable lines of code, whilst miss refers to the number of executable lines that is not run by the tests.

7.3 Single-user performance tests

To find flaws in the user-friendliness of the project, we used a single-user performance test. We asked an independent person who had never used the product before to test out our site. This test revealed that the event-form was very unintuitive. The test subject thought the time they entered was the final time, not an interval. As a result of this test, we changed the layout of the create event-form. We used to have a big form where the user would input all the information on a single page. Instead, we implemented a form where a user inputs one and one data point. This way, we can put a more descriptive instruction to the user for each input without overwhelming the user with text.

8 Discussion

8.1 Justifying data collection

The data that is collected when creating an event is collected to describe the event for any invited participant, to give them the information they need to decide if they want to attend or not. Schedge does not collect any other data points than what the user explicitly enters, as the users should have full control over what information the service has about them. The time slot information is collected and used to calculate the best time to host the event, so as many participants as possible can attend. Information about who is friends with whom is collected because each account has an overview of their friends on the dashboard. The collection is done to visibly show a user their friends, as well as give them an easy way to invite friends to upcoming events.

8.2 Practicality of the Riise-Hofsøy algorithm

As mentioned the Riise-Hofsøy algorithm has a complexity of $O(N^2)$. However, in any practical use case of this algorithm, the computational cost will be negligible. Schedge as a product is mainly aimed at small events with few participants. It is also highly unlikely that there will be a matryoshka doll effect on all-time slots committed, and the likeliness of this effect occurring will decrease as the number of time slots increases. Therefore, the likeliness of a worst-case complexity is low. For larger events where there will be a large number of participants, it doesn't make sense to use our product. The more participants there are in the event, the less likely it is to find a time slot that fits for everyone, in these cases setting a hard date and forcing participants to make time is a better solution.

8.3 Benefits/disadvantages of the agile methods used

Some agile methods that were diligently used for the project are test-driven development, pair-programming, incremental planning, collective ownership and continuous integration.

8.3.1 Test-driven development

As mentioned above in the software development process section, test-driven development was further customized to fit our needs during the development phase. By altering the test-driven development to parallel test-driven development, instead of test-first development, the team could both make the development of the features faster and also catch needed function changes before all the tests were implemented. This led to less of a need to refactor the existing code-base, as the tests are written at the same time as the code is being implemented. This means that overall less time is spent on each feature.

8.3.2 Pair-programming

Another method that was frequently used throughout the project is pair-programming. This method has both its advantages and disadvantages. As the team was new to the Agile development process, pair-programming helped with getting the team to a common understanding of how to structure and implement the different features of the project. The pair-programming sessions lead to exchanging of ideas and methods that helped speed up the implementation of features at the start of the project. As the team's knowledge about how to work an agile procedure increased, the need for pair-programming decreased. This is because pair-programming is essentially two people working a one man's job, which would halt progress as the developers got to know how to work on the project. On top of that, when two different developers work on a shared feature, their opinions on how the feature is best implemented will most likely vary. This would cause the time to be spent just getting to a common ground before resuming implementation.

8.3.3 Incremental planning

A benefit of incremental planning is that it makes the product more viable for changes and the development process more flexible. This is because there is no “grand plan” that needs to be followed thoroughly, but the team could instead modify its plans as it is needed during development. This was done within our development process as we changed from a calendar-based idea to an event scheduling software based on user suggestions. The reasoning behind this change is that few users have their whole schedule uploaded onto the web.

8.3.4 Collective ownership

Having the team take collective ownership over the product ensures that no team members are slacking off, as everyone has equal responsibility of the product getting to a “potentially shippable product” for the end of the development process. This will lead to a more equal workload for every team member because there is no need for the team to pick up another developers workload.

8.3.5 Continuous integration

Using continuous integration in the developing process assists developers in developing different functionality at the same time. Developers need only think about their functionality whilst developing and does not need to concern themselves with being compatible with others functionality. Continuous integration also gives the developers constant access to the latest version and latest functionalities of the project. The optionality of documentation in the continuous integration pipeline helps developers have a complete focus on implementing the functionality without having to waste time on docstrings and comments. Documentation is still an important part of the project but does not necessarily have to be done at the same time as the functionality is being written. Instead, the developer can look back at the functionality and use the documentation report to see what needs to be documented.

8.4 What we have learned

8.4.1 Proper CI/CD:

As for this specific project, there was some limited knowledge of the possibilities behind continuous integration at the beginning. The specific functionality of continuous integration as is described earlier in the report were not used until halfway into the project. For this project to reap the full benefits of continuous integration, this should have been implemented at the very beginning of the project. This would lead to better test coverage statistics at the end.

8.4.2 Proper implementation of tests

The importance of writing enough and good tests with proper coverage was discovered early on in the project. This helped writing new functionality without compromising the rest of the code base. As with the continuous integration, proper testing was not done in the beginning. This lead to a team member having to write a lot of tests in retrospect.

8.4.3 Development tools

During the project the power of Agile as a development tool became clear. It was easy to keep track of the current state of the project whilst new features were implemented rapidly and developers found it motivating to be able to see the progress of the product between the sprints. The practices used helped the development process run smoothly during this project and no developer was left without something to do.

8.5 Alternative methods that could have been used

8.5.1 Sub-sprints within sprints

By dividing sprints into sub-sprints, we could get an even better overview of the project progress. This would help the developers further with the development as they would have a clearer understanding of how much workload different features will require. In addition this would help the teams synchronise the development of new features, and balance the workload on features within different sub-sprints.

8.5.2 Waterfall development

Waterfall is a developing tool, unlike Agile, that requires a linear developing process where each stage revolves around the result of the preceding stage. This makes for a less flexible design process. Waterfall would lead to a more defined end goal in the beginning which could lead to less misunderstandings between developers. However, it is less equipped for sudden changes in the product, which could arise from a multitude of factors. This would not fit us, because the end goal of the project changed dynamically during the sprints. This led to a more refined end product. If we had used waterfall we would have been forced to start the development process all over again, for each change.

8.5.3 Increased use of user tests

The codebase is thoroughly tested from a functional standpoint. However, more user testing should have been applied earlier in the project to gather more data about the user-friendliness of Schedge. This would have prevented the need for refactoring of the whole user interface as the developers would have known how the website is perceived from an outside perspective.

8.6 Future plans

8.6.1 Mail server

Setting up a mail server will allow communication with users of the application. Right now the only use for a mail server would be to send an email with a verification link when a new user has just signed up or when a user has forgotten their password and would like a link to reset their password.

8.6.2 Chat function

By having a chat function, users can easily communicate with other users in the event. The chat could be automatically created for each event. With a chat feature, the attendees of an event could plan what to bring or what to eat without having to rely on another app.

8.6.3 Picture feed

During the event, users may have taken pictures of the group or something else related to the event. The picture feed will be a place to gather these images so all those who participated can view and download them as they like.

8.6.4 Integrated calendar

The current UI, for submitting the users available time slots is not intuitive to everyone. Having a calendar where the user could click and drag to receive some visual feedback would hopefully help with this. The calendar can also synchronize with the user's calendar, allowing the user to see where they have other plans. The user's submitted time slots and the time of the event will also be synchronized to the user's calendar app, making it more convenient to keep track.

8.6.5 Integrated map

Users can already select a location to have their event at, but with a map, this could also be visualized by showing directions from the user's current position. The user should also be able to open the directions in their google maps application to actively get directions as they are heading towards the destination.

8.6.6 Better user profiles/friends system

Although this app doesn't rely too much on having huge user profiles, having a small profile for each user where they can have a profile picture and such can hopefully make the experience a little more personal and encourage adding friends to have a more interactive environment.

8.6.7 Business facing

Right now, the application is focused on casual events among friends and family, but it can also function as a great meeting scheduler. Adding an option for creating more formal events where some features, like pictures, are disabled, and others, like a link to a Zoom meeting, are added. Also having an invite link instead so the

event can be attached to a mail and sent out easily to coworkers and colleagues without having to know their username or have them as a friend. The link feature works in a more casual event scenario too.

8.6.8 Extend the duration of an event

Currently, an event can only last for up to 24 hours, which means planning a weekend fishing trip is rather difficult. The plan is to allow events of arbitrary length. The reason longer events are difficult to implement is that having the user set a start and end time of the event does not make sense if the event spans several days. Therefore, we plan to make the user choose between a multi-day event or a regular event when they create it, and change how the time slots are selected from that choice. A user would instead select whole days where they can attend.

8.6.9 Automated data cleaning

To comply with GDPR, data that is no longer used or needed has to be removed. As that job is currently done manually, this needs to be automated in the future. This entails system checks that look for data and users that have not been active for some time, and remove that data from the database. If the system notices users that have been inactive for some number of months, an email and notification should be sent. This should be done to inform the user about what will happen if they continue to be inactive, giving them time and the opportunity to decide what to do about their profile and data.

8.7 Results

The result is a working web application. It has enough features for it to be usable as a scheduler for groups wanting to host and determine when events should be held. The host can invite other users to their events and together they can decide when it suits best to hold the event. Each user may host or attend as many events as they wish, and can even add friends.

9 Conclusion

This report went into detail and described the agile development process, implementation and use of a web-based product for scheduling events. It explained what different agile methods were used for this project, such as continuous integration, pair programming, collective ownership, test-driven development and incremental planning. It also describes how the process of implementation was done, what measures and features were included in the project and how the end functionality of the product works. The result is a web application that can be used for scheduling and deciding when to host an event with all participants' best interests in mind.

10 Acknowledgements

We would like to thank our TA and Scrum Master, Owais Qayyum, for help and guidance this semester.

References

- Doodle (2021). *The Doodle Story*. <https://doodle.com/en/about-us/>.
- European Commission (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)* (GPDR). <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- Leonel, Alvaro et al. (2020). *django-notifications*. <https://github.com/django-notifications/django-notifications>.
- Marzullo, Keith Ansel (1984). “Maintaining the time in a distributed system: an example of a loosely-coupled distributed service (synchronization, fault-tolerance, debugging)” (Maintaining the time in a distributed system). PhD thesis. Stanford University.
- Meyer, Robinson (2018). *The Cambridge Analytica Scandal, in Three Paragraphs*. <https://www.theatlantic.com/technology/archive/2018/03/the-cambridge-analytica-scandal-in-three-paragraphs/556046/>.
- Microsoft (2021). *Outlook*. <https://outlook.live.com/owa/>.
- Mozilla (2021). *Ajax*. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>.
- Shneiderman, Ben and Catherine Plaisant (2004). *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. 4th ed. Pearson Addison Wesley. ISBN: 0321197860.
- Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Pearson. ISBN: 9780135210642.

Appendix A

Feature list

Table 2: Feature/contributor table, B represents backend, F front-end, T for testing, X misc.

Feature\Name	Elias	Vetle	Theodor	Iris H	Marte	Alexander
Riise Hofsøy algorithm	B	T				
Friends	B	T	F		F	
Authentication	B	B				
GDPR						BF
Homepage					F	
My page/dashboard			B		BF	BT
About us			F		F	F
Sign in	B	B			F	
Sign up	B	B			F	T
Forgotten password	B	B			F	
Terms & conditions						BF
Create event			BT	BT	F	
Event			BT	BT	F	T
Edit event			BT	B	F	T
Delete event			B	BT		
Create time slot	B		BT	BT	F	
Delete time slot		T	B	B		
Merge potential time slots	B	T				
Select time slot			BF	BF	F	T
Invite participant			BT	BT	F	T
Remove participant			BT	BT		T
Participant leave event			BT	BT		T
Notifications	B		BT	BT		T
Accept invite through inbox					F	
Silent notification			B			
Accept invite through notification			F			
Attendees of an event			B	B	F	
Time slot validation	B		BT			
Add friends	B	T	F			
Accept/reject friend request	B	T	F		F	
CI		B				B
Splitting time slot			B	B	F	
Delete user						BTF
Final time slot			B	B		
Logo						F
Documentation	X					X
README instructions	X					

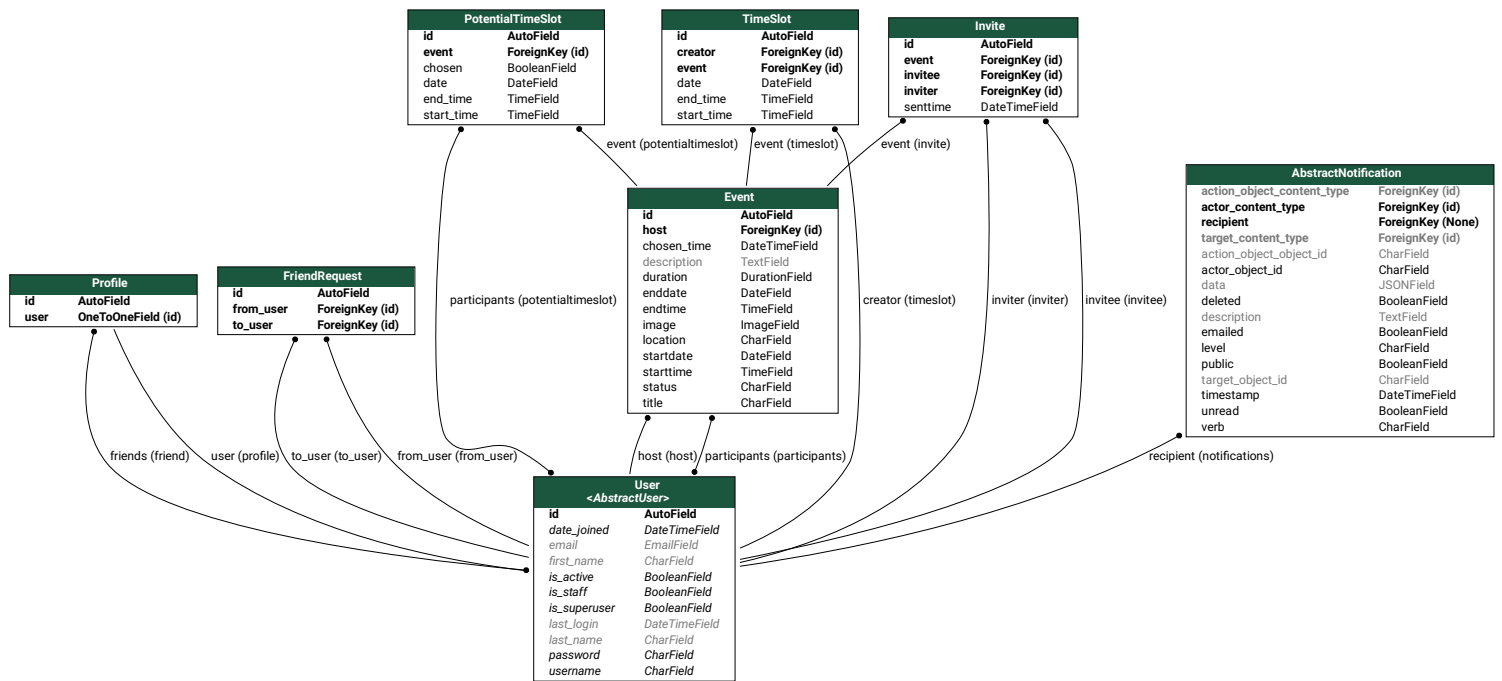


Figure 5: Entity Relationship diagram of the database used for schedge

Appendix B

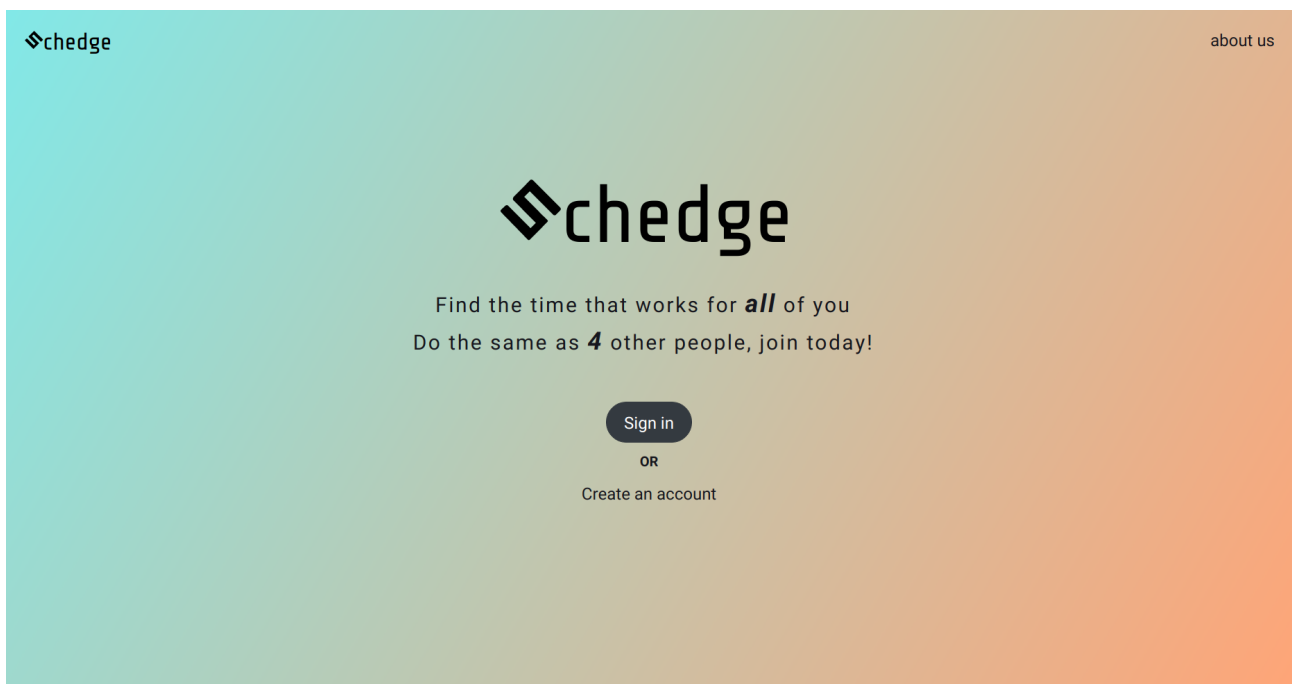



Figure 6: Landing Page


about us

chedge

Did you forget your password? [Reset Password](#)

Don't have an account yet?

Figure 7: Sign In Page

about us

Create a chedge Account

☐ By clicking Sign up, you agree to our [Terms](#).

Do you already have an account?

Figure 8: Sign up Page

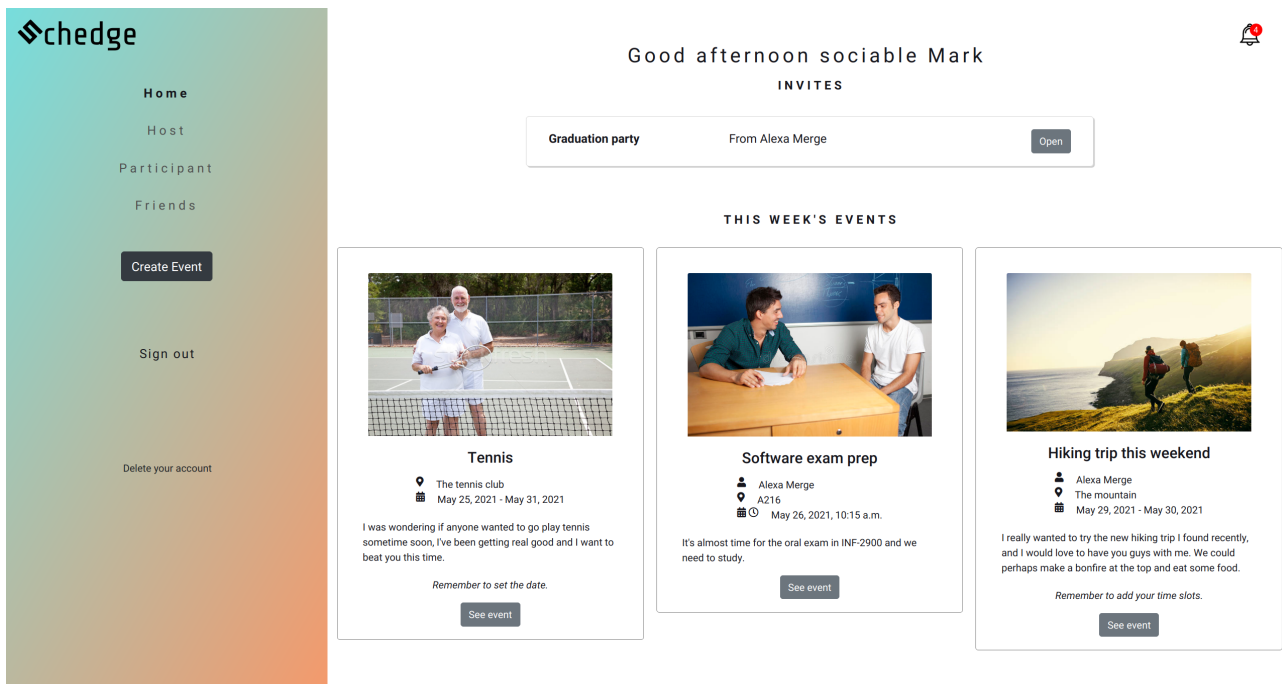


Figure 9: My Page

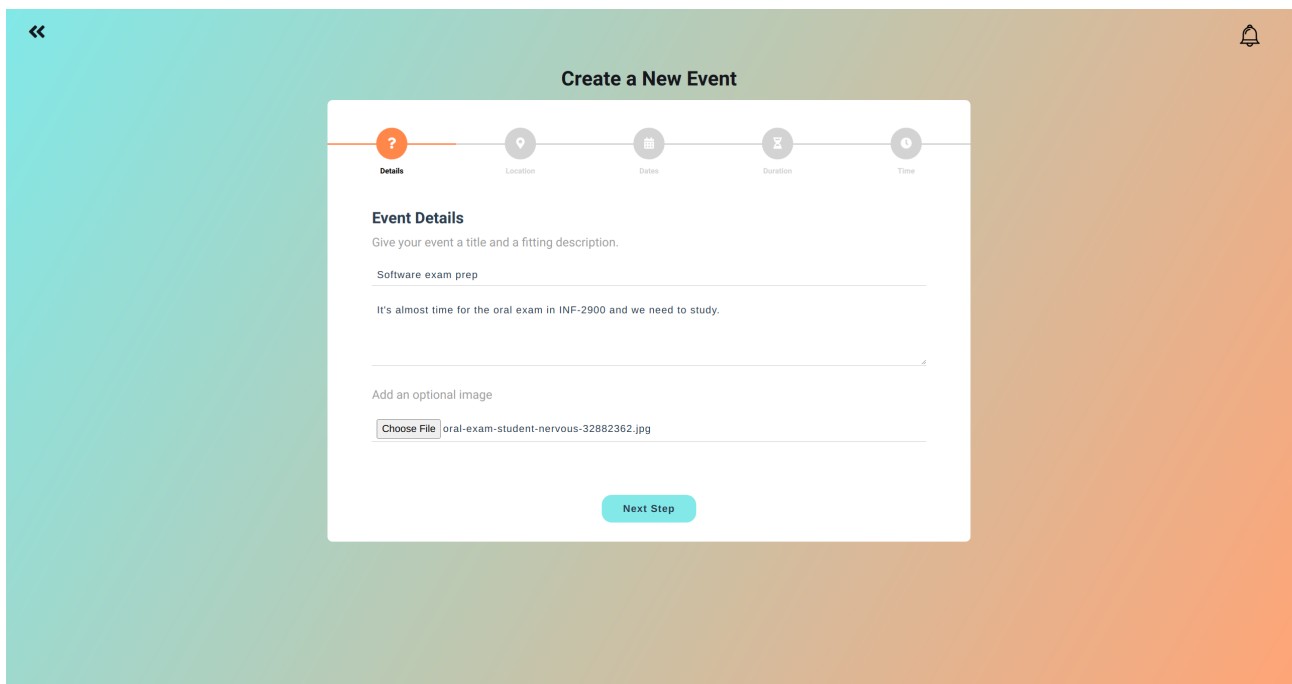


Figure 10: The Create an Event page

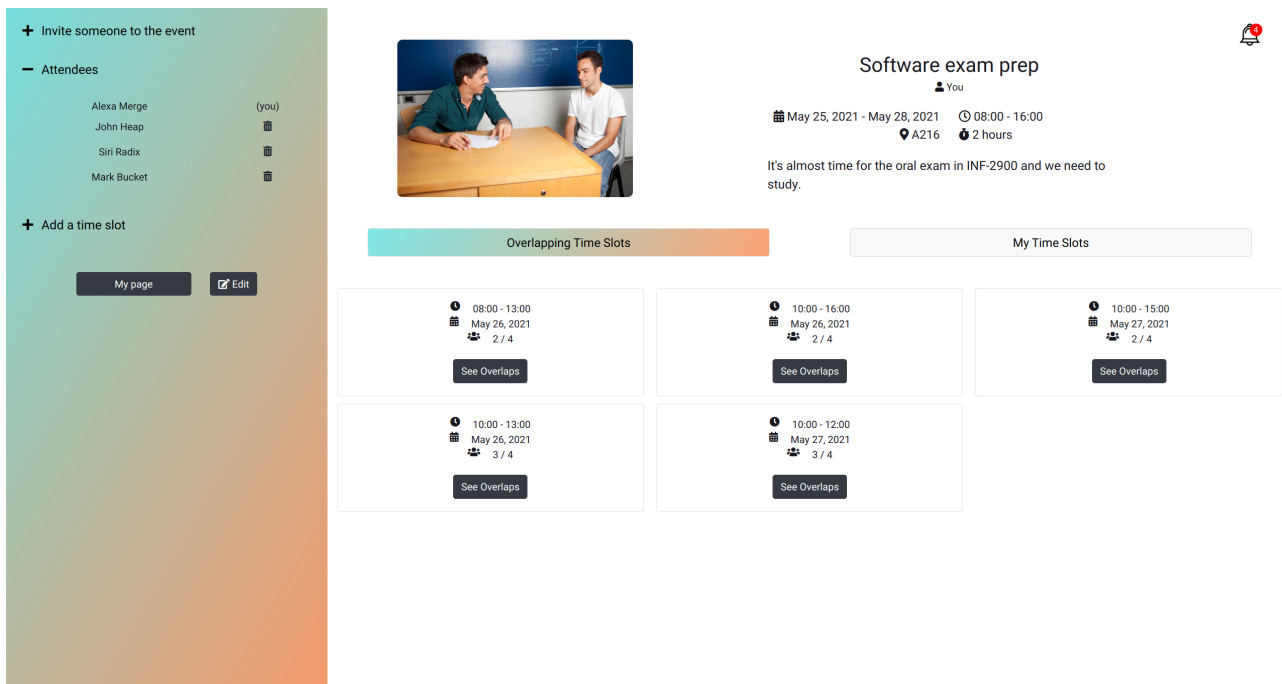


Figure 11: Event page, showing the overlapping time slots

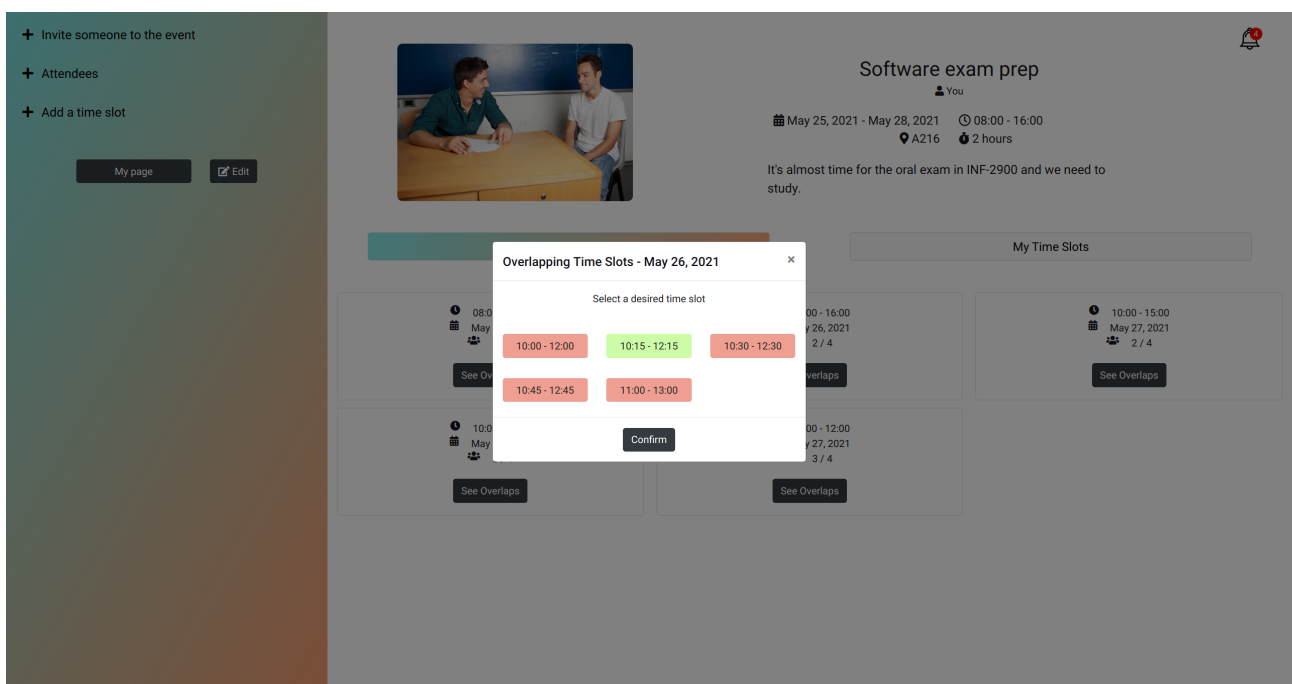


Figure 12: Event page, determining the time slot

User stories

- **User story 1:**
As an attendee, i want to be able to leave an event at any given time.
- **User story 2:**
As a host, i want to be able to un-invite someone from an event that i am hosting if i no longer want the person to attend.
- **User story 3:**
As a host, I want to edit my event.

- **User story 4:**
As a user, i want to get notifications when i am invited to an event.
- **User story 5:**
As a user i want the option to remove a pending friend request if i accidentally added the wrong user as a friend.
- **User story 6:**
As a host, i want the option to delete my event if i no longer want to host it.
- **User story 7:**
As a host I want to see the events I have created so I know the details about the event, such as the participants, length, time etc.
- **User story 8:**
As a host I want to see how many have answered the event request, so I have an idea of when I can determine the exact time.
- **User story 9:**
As a host I want to be able to change the details of my events in case of unforeseen situations.
- **User story 10:**
As a host I want to be able to delete the event if I no longer have time for it.
- **User story 11:**
As a host I want the option to look at the results and determine the time when I want to.
- **User story 12:**
As a user I want to see the events that are coming up.
- **User story 13:**
As a participant I want to see the events I am participating in where the exact time is yet to be determined.
- **User story 14:**
As a participant I want to see the event invitations I get.
- **User story 15:**
As a participant I want to answer the invitations.
- **User story 16:**
As a participant I want to see the invitations I get to events.
- **User story 17:**
As a participant I want to answer the invitations.
- **User story 18:**
As a host, I want to invite guests.
- **User story 19:**
As a host, I want to edit the event.
- **User story 20:**
As a user, I want to add time slots that i am available.
- **User story 21:**
As a user, I want to log in to my own private account.
- **User story 22:**
As a new user, I want to be able to register an account.
- **User story 23:**
As a user, I want to see what the web site is for so that I know if it fits my purpose.
- **User story 24:**
As an existing user, I want to receive an email where i can reset my password.

- **User story 25:**
As a host i want to exclude times that does not work for me, for example at night or on weekdays.
- **User story 26:**
As a host I can add details about my event so my guests know what the event is. This includes a description, the location, time and possibly a picture.
- **User story 27:**
As a host I can send invitations to friends, and the app will find the best fitting time for everyone to host the event.
- **User story 28:**
As a host, I can specify how long the event I plan will take, to find a fitting spot.
- **User story 29:**
As the creator of an event, I want to specify that lunch is between 9am and 2pm.
- **User story 30:**
As a user i want to be informed about what data is collected and explicit reject or accept it.

Product backlog

List of the features developed in the different sprints

Sprint 1 backlog

- login button
- sign up button
- sign out button
- register user
- dashboard/homepage
- create event
- event information form
- event page
- add time slot
- edit event
- delete event button
- current location button

Sprint 2 backlog

- front page
- welcome message
- notifications
- notification bell
- accepting/rejecting event via notification
- invitations section*
- list of links to upcoming events
- list of links to events that user is a participant in
- remove friends from event
- back to dashboard button

Sprint 3 backlog

- delete account button
- friend page
- friend list
- user search bar for adding friends
- add user as friend
- delete friend
- invite friends to event
- accepting/rejecting friend request via notification
- Delete user with confirmation
- select time slot
- about us button
- about us page
- twitter button on about us page
- Terms and Conditions page
- Checkbox to agree terms before sign-up
- host page
- list of links to own undecided events
- list of links to own decided events

Meeting logs

03.02.2021 - Group meeting:

MVP (Minimum viable product):

Start page:

- Logo
- About us
- Information about what the service is.
- Sign in button
- Sign up
- Create event button

My page:

- Log in
- Name, id, password, mail
- Show upcoming events
- create event

Event:

- Create
- Edit event
- iD, Name, description, timeslot, location
- Send invite
- Accept/reject request
 - Accept:
 - Add and respond with available timeslots.
 - Use callendar (upload) to generate available timeslots.
 - choose which of the generated timeslots to include in responce.
- Accept/reject invite

Internal teams:

- Vetle & Elias
 - Setup database
 - Start page
 - Sign in
 - Sign up
- Alex & Marte:
 - My page
 - Send requests
 - Send invite
- Iris & Theodor:
 - Event
 - Create event

09.02.2021 - Group meeting:

- The team discusses about the project.
- Asked about pipeline, tests and CI:
- Owais informed us about Udemy (free of charge)

09.02.2021 - Meeting between Marte & Alex:

- Discuss implementation plans to be executed during next work session (which will be thursday 11.02.2021).
- Archive:
 - -Move past events to archive DB so they are not considered when querying for new/upcoming events
- My events:
 - User ID = HOST:
 - * Test cases:
 - * Get 'my events' if userID matches.
 - * (Future: include status - number of answers, edit event, terminate requests to conclude number/which participants.)
- Upcoming:
 - Host: Upcoming event that userID is hosting.

- Participant: Upcoming events that userID is a participant of. (Has accepted invitation)
 - Test cases: Get 'upcoming events' where userID is either host or participant.
- Event:
 - Include GroupID
- Group:
 - Create 'group' table to include participants with:
 - * userID
 - * groupID
 - * eventID
 - * hostID (?)
 - * If a participant is in several groups a new entry of userID with the different groupID is created.
- TODO:
 - add userID to event model.
 - Testcase
 - Create group model
 - Upcoming events - query host
 - Upcoming events - query participant

11.02.2021 - Meeting between Marte & Alex:

- Discuss user interface on mypage (aka profile page):
 - 3 column layout
 - left:
 - * Upcoming events as HOST, not concluded yet
 - middle (upcoming events):
 - * Upper - Upcoming events as HOST, concluded.
 - * Lower - Upcoming evnts as participant.
 - right:
 - * upper - Request (like invite but available times included in respons)
 - * lower - invitaions, a time and date has been set and the user is invited.
- Goal for iteration 1:
 - Tests for any functionality implemented. (tests written first, implement later).
 - Functional prototype of mypage that is responsive using a dummy-database (if the real db is not up and running yet.).

11.02.2021 - Marte & Alex - agenda:

- Expand database with userID (although not real):
 - ————— with status (Concluded, unresolved)
- Mypage - ONLY own events.
- Upcoming events HOST:
 - Only userID as host events.
- Upcoming events participant:
 - Upcoming evnts where user is NOT host. (for now, to simulate.).
- Write template to match discussed interface better.
 - Starting with two collumns.

15.02.2021 - Meeting between Marte & Alex:

- Status update:
 - The site runs with template and html.
 - * Interface without backend
 - backend is incomplete.
- TODO:
 - Write tests
 - Complete backend.

16.02.2021 - Group meeting:

- Physical or digital meetings moving forward:
 - Seems like physical is the way to go
 - Will be physical meetings from next week on.
- Theodor is showing what he and Iris has done on event page.

18.02.2021 - Meeting between Marte & Alex & Owais:

- Problem:
 - doing subquery to get every event connected to a user, but not where the user is host.
- proposed solution:
 - Doing if else test in the template to filter user!=host

18.02.2021 - Marte & Alex - agenda:

- Fix my page views
 - Implement filter in template to get every event where user is not host.
 - Tests.

23.02.2021 - Group meeting:

- - Add field to group table:
 - Host = True/False
- Check 'token' if user is authenticated and which user is logged in.
- Agree on and create a base for a coherent design throughout the site.
 - Change blocks in base to customize for each url.

02.03.2021 - Group meeting:

- Shown demo, css is a bit broken for some pages.
- For the next iteration:
 - Do more calculations on when would be the best time.
 - Write more tests

08.03.2021 - Group meeting:

- Meeting about what to do during the next sprint/iteration:
- Attendense:
 - Elias
 - Alex
 - Iris
 - Vetle
- TODO:
 - Tests:
 - * mypage - Alex/Marte
 - * home/reg/sign - Elias/Vetle
 - CI (Continuous integration)
 - Fix HTML to work with base HTML and css.
 - * Make more generell (MARTE - PLEASE HELP)
 - Invite people to event.
 - mypage - see only events for given user.
- Next features:
 - nly host can edit events.
 - Invite friends.
 - See timeslots that works for x-amount of people.
 - * Determine time for event.